MASTER EN SCIENCES INFORMATIQUES

Spécifications d'un prototype d'interface utilisateur pour une base de données de documentation selon une approche orientée objets

Godfroid, Thierry; Plancq, Laurent

Award date:
1992

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Namur

Institut d'informatique

Spécifications d'un prototype d'interface utilisateur pour une base de données de documentation selon une approche orientée objets

Thierry GODFROID
Laurent PLANCQ

Mémoire présenté en vue de
l'obtention du diplôme de licencié et maître en informatique

Promoteur : Madame le professeur Monique Noirhomme-Fraiture

Année académique 1991-1992

Résumé

Le Centre for Educational Sociology de l'Université d'Edimbourg réalise depuis de nombreuses années des enquêtes par questionnaires centrées sur le thème de l'éducation. Elles ont généré une quantité remarquable de documentation que le CES concentre dans une base de données. Notre mémoire répond au problème de l'accès à cette base, en décrivant et spécifiant (selon une approche orientée objets) un prototype d'interface utilisateur et en montrant comment il peut être aisément modifié.

Abstract

For numerous years the Centre for Educational Sociology of the University of Edinburgh carries out questionnaire-based surveys about education. These surveys produce a large amount of documentation data that CES is concentrating into a database. Our thesis answers the problem of access to this database by describing and specifying (according to an object-oriented approach) a prototype of user interface and by showing how it can easily be adapted.

Remerciements

Tous nos remerciements à Madame Monique Noirhomme-Fraiture qui a pris en charge la direction de ce mémoire et nous a soutenus dans notre travail.

Nous tenons également à adresser nos plus vifs remerciements à Madame Joanne Lamb pour son accueil chaleureux et ses conseils toujours judicieux.

Nous remercions Monsieur Eric Dubois pour sa disponibilité et son aide précieuse.

Nous adressons un merci tout particulier à Monsieur Jean Vanderdonckt pour la pertinence de ses remarques.

Il nous faut aussi remercier les membres du CES qui ont été disponibles et accueillants tout au long de notre stage. Leur gentillesse est pour beaucoup dans le bon déroulement de notre stage.

Un grand merci enfin à toutes les personnes qui ont participé, d'une manière ou d'une autre, à l'élaboration de ce travail.

Table des matières

Introduction

De nos jours, l'information prend une place de plus en plus importante. Une partie de notre travail quotidien consiste à rechercher les informations nécessaires à la tâche qui nous est confiée.

L'information est abondante et complexe, des moyens doivent être mis en oeuvre pour nous permettre de l'utiliser de manière simple et efficace.

L'informatique joue un rôle important par rapport à ces deux aspects : elle fournit en effet le moyen de traiter efficacement des quantités importantes de données.

C'est dans cette perspective que travaille le Centre of Educational Sociology de l'Université d'Edimbourg. Ce centre réalise de grandes enquêtes postales qui génèrent une quantité impressionnante d'informations que les chercheurs vont analyser pour aider les autorités régionales et locales à établir leur politique en matière d'éducation. Puisqu'elle dépeint la réalité, cette information se structure d'une manière complexe.

Un point crucial de cette analyse est l'accès à la documentation sur ces informations. Dans ce domaine, le CES a une approche originale, qui consiste à concentrer les informations sur les enquêtes (également appelées métadonnées) dans une base de données réalisée à cet effet. Le but avoué est, à terme, de remplacer les moyens actuels de documentation (dictionnaire papier notamment) par un système informatique complet.

Notre participation à ce projet à été de réaliser un prototype d'interface utilisateur permettant aux chercheurs d'accéder à ces métadonnées, d'une manière intuitive et rapide. Le présent travail décrit pourquoi et comment le prototype a été réalisé et propose une approche de spécification indépendante de tout support physique et logique et aisément modulable au fil du temps.

Dans une première partie, nous approchons le problème de manière empirique. Cela se fait au travers d'une description du contexte de notre stage ainsi que du travail que nous y avons réalisé. Nous attachons ensuite une attention particulière à l'analyse du résultat obtenu et de la façon dont il a été perçu.

La seconde partie a comme but de jeter les bases théoriques nécessaires pour aborder la phase de spécification. Nous débutons en présentant les concepts issus de la théorie de l'ingénierie logicielle, avant de choisir et de justifier une approche particulière. La présentation du langage qui la supporte conclut cette partie.

La troisième et dernière partie se compose principalement des spécifications du prototype que nous avons réalisé. Nous présentons également la méthodologie que nous avons adoptée. Enfin, nous montrons comment adapter ces spécifications à de nouveaux besoins.

Première Partie

Bases empiriques : un prototype d'interface

Cette partie de notre travail, qui favorise une approche plus pragmatique de la conception de l'interface à une base de données de documentation, est en fait très largement basée sur notre collaboration avec le Centre for Educationnal Sociology de l'université d'Edimbourg, où nous avons réalisé notre stage.

Quatre chapitres forment l'ensemble de cette partie et retracent l'évolution de notre travail. Celui-ci est principalement lié à l'expérience en métadonnées du CES. C'est pourquoi nous étudierons tout d'abord plus en détail les caractéristiques et les réalisations de ce centre de recherche, avant d'aborder, en trois phases, la conception du prototype d'interface utilisateur. Ces phases seront les suivantes : une description approfondie de l'élaboration du prototype, une analyse de celui-ci en termes de comportement et d'ergonomie, et enfin, une présentation succinte d'une première réaction des utilisateurs.

Chapitre 1

Le cadre de travail

Dans ce chapitre, nous présentons le contexte dans lequel se sont déroulés notre stage et notre travail. L'objectif est de familiariser le lecteur avec les différents concepts que nous utiliserons au cours de notre exposé.

1.1 Le Centre for Educational Sociology (CES)

Fondé en 1972, le CES est un centre de recherche de l'Université d'Edimbourg. Il est principalement financé par cette dernière, le département écossais de l'éducation (Scottish Education Department - SED) et le conseil de recherche économique et social (Economic and Social Research Council - ESRC). Il existe également d'autres sources de financement plus ponctuelles comme d'autres départements écossais, des autorités locales et des fondations privées.

Le CES conduit des recherches dans le domaine de l'éducation, de la formation, du marché de l'emploi, des jeunes et de leur contexte familial, des changements socio-économiques, des politiques gouvernementales dans ces domaines et de leurs conséquences.

Les données statistiques sur ces sujets sont récoltées via des enquêtes postales auprès des jeunes écossais. Le personnel du CES peut être vu comme un ensemble de quatre sous-groupes, chacun remplissant une fonction bien précise dans le processus de l'enquête : les chercheurs, l'unité informatique, l'unité d'administration des enquêtes et le support administratif.

L'enquête se déroule de la façon suivante. En fonction des besoins et demandes des divers commanditaires, les chercheurs déterminent quelles sont les questions qu'ils désirent voir posées aux destinataires de l'enquête. Via un logiciel réalisé par l'unité informatique, l'unité d'administration des enquêtes confectionne le questionnaire à partir de ces questions. Grâce à une base de données des adresses des

individus maintenue à jour par le support administratif, l'équipe d'administration des enquêtes envoie les questionnaires à tous les membres de l'échantillon retenu pour cette enquête. En collaboration avec cette équipe, les informaticiens préparent les écrans de saisie destinés à l'encodage et la base de données prévue pour le stockage des résultats. Les questionnaires complétés sont retournés à l'administration des enquêtes qui encode les réponses. Notons que pour augmenter le taux de réponse, le support administratif envoie des rappels aux personnes dont les questionnaires sont restés sans réponse et retrouvent la trace des individus ayant changé d'adresse. Une fois l'encodage terminé, l'unité informatique procède à un traitement des données brutes pour les mettre à disposition des chercheurs. Ceux-ci, éventuellement avec l'aide des programmeurs, réalisent des traitements sur les données pour obtenir les informations pertinentes à leur travail.

Soulignons enfin le contexte mouvant du CES : ses membres ont rassemblé sur une période de temps considérable un ensemble complexe de données interdépendantes. La durée de cette période a pour conséquence que se sont produits des changements dans la réalité et que les questions ont dû être modifiées en parallèle. Notons que cette longue durée implique également des changements dans la technologie de collecte et de support des données.

1.2 L'unité informatique et ses réalisations

Nous allons voir quels sont les différents projets qui sont ou ont été menés par les informaticiens du CES.

Questmast est un logiciel conçu par les programmeurs pour aider les chercheurs à mettre sur pied les questionnaires nécessaires aux enquêtes postales. Questmast dispose essentiellement de deux fonctions : la mise en page d'un questionnaire et la génération d'un schéma de base de données. Avec la première, le programmeur, sur base de la définition des questions fournies par le chercheur, procède à leur mise en forme sur une page de questionnaire. La seconde fonction produit les éléments nécessaires à la construction du schéma de la base de données qui servira lors de la phase d'encodage. Questmast génère d'autre fichiers desquels certaines données de documentation peuvent être extraites, par exemple les commentaires des chercheurs lors de la conception des questionnaires.

A partir des fichiers fournis par Questmast, les programmeurs conçoivent les bases de données ainsi que les écrans de saisie utilisés pour l'encodage.

Les programmeurs développent des procédures qui servent, une fois la base de données complétée, à traiter les données brutes pour qu'elles soient utilisables par le progiciel statistique SPSS^X. Ce dernier est l'outil utilisé par les chercheurs pour analyser, comparer l'ensemble des données à leur disposition.

En plus de ces différentes tâches, l'unité informatique s'est investie dans un travail de documentation des données relatives au processus d'enquête, le projet Docdb.

1.3 DOCDB

Le nombre et la taille des enquêtes ont pour conséquence l'obligation de manipuler un volume important d'informations. Pour pouvoir utiliser efficacement celles-ci, il est nécessaire de les documenter.

Le projet Docdb consiste à concevoir et implémenter une base de données de documentation. "Une base de données de documentation est un dictionnaire de données prévu pour organiser et gérer des métadonnées (c'est-à-dire des données à propos de données) et des données documentaires. Elle facilitera la mise à disposition de documentation sur les données à analyser. (...) L'implémentation d'une base de données de documentation permettra aux chercheurs de consulter une seule source de documentation pour différentes études (...)."[1]

Le projet comporte en fait deux volets : procéder au remplissage de la base de données dont l'objectif est de fournir une documentation exhaustive et mettre à la disposition des utilisateurs un outil pour accéder aux informations.

Jusqu'à présent, la documentation est disponible sous la forme de dictionnaires sur papier qui reprennent pour chaque dataset la description des variables qu'il contient. Signalons que par dataset, le CES entend toute collection structurée d'informations, qu'il s'agisse d'une base de données, d'un fichier de données formaté pour SPSS^X ou de tout autre structure.

Notre travail est une première étape dans la mise à disposition de cette documentation grâce à une interface utilisateur en ligne.

---

[1] [Ritchie91, appendice 1, page 1

1.4 Support physique et logique du CES

Nous terminerons ce premier chapitre en présentant l'environnement informatique du CES sur lequel nous avons développé notre travail.

L'université d'Edimbourg met à la disposition du CES les moyens informatiques nécessaires à son fonctionnement. Cela comprend l'accès au réseau et aux mainframes de l'université, ainsi qu'au support utilisateur. Chaque programmeur et chercheur dispose soit d'un terminal texte, soit d'un PC équipé de façon à pouvoir se connecter aux mainframes. Notons de plus que le CES vient de se doter d'un réseau local et d'un serveur de fichiers. A terme, une partie de leurs applications seront disponibles sur ce support.

La plupart des bases de données sont gérées par le progiciel SIR (Scientific Information Retrieval). Celui-ci est composé de quatre modules.

Le premier, appelé SIR/SCHEMA, permet de définir et de modifier de manière interactive le schéma d'une base de données.

Le module SIR/DBMS est utilisé pour créer une base de données à partir de ce schéma. De plus, ce module supporte le langage PQL (Procedural Query Language) qui permet de réaliser des programmes sur les bases de données.

Un troisième module, SIR/FORMS, fournit à l'utilisateur des écrans simples pour consulter et modifier aisément le contenu de la base de données.

Le quatrième module, nommé SIR/SQL+, permet d'utiliser SQL dans l'environnement SIR.

Revenons maintenant à PQL. C'est un langage fonctionnel qui offre de nombreux moyens d'accès aux bases de données et de manipulations de chaînes de caractères et d'écrans. Il est syntaxiquement proche du COBOL[2].

---

[2] L'annexe 3 contient un exemple de programme PQL.

PQL souffre des limitations suivantes :

- impossibilité de l'utiliser sans une base de données courante (même vide);
- pas de gestion de variables dynamiques (notion de pointeur);
- pas de réel appel de procédures (cependant, certaines procédures peuvent être regroupées dans différentes unités appelées "familles". La fonction CALL dans le programme appelant est alors en réalité une copie de texte plutôt qu'un appel).

Notons pour finir qu'en dépit de ces faiblesses, PQL dispose de certaines fonctionnalités astucieuses comme la possibilité de mémoriser et récupérer des portions d'écran ou la manipulation aisée des index.

Chapitre 2

L'élaboration du prototype d'interface utilisateur

Maintenant que nous avons défini, tant au niveau organisationnel que technique, le cadre dans lequel va se réaliser notre travail, nous allons passer à l'analyse de celui-ci.

Celle-ci comporte plusieurs étapes retraçant l'évolution du projet dans le temps. Premièrement, nous justifions notre approche du problème. Ensuite, vient l'analyse proprement dite. Deux phases ont été réalisées en parallèles : la détermination des besoins des utilisateurs et l'étude de l'existant, à savoir la base de données. En nous basant sur ces prémisses, nous donnons enfin une forme à notre projet, en définissant l'architecture de l'application et la spécification des accès à la base de données.

2.1 Pourquoi un prototype d'interface ?

Cette question peut en fait se décomposer en deux parties. Nous pouvons d'abord nous demander pourquoi le CES a besoin d'une application d'interface utilisateur dans le cadre de Docdb. Nous pouvons ensuite nous demander pourquoi le développement de cette interface passe par la réalisation d'un prototype.

2.1.1 Pourquoi une interface utilisateur ?

"Une interface utilisateur est un système informatique utilisé par une personne pour réaliser une tâche accomplie à l'aide de moyens informatiques"[1].

Le but d'une interface est de permettre à l'utilisateur d'accomplir le mieux possible la tâche pour laquelle cette application a été concue.

---

[1] [Bodart89]

Dans notre cas particulier, il s'agit donc de créer une interface qui permette à l'utilisateur de consulter au mieux la base de données Docdb. Sans cette interface, Docdb perdrait tout son sens car nous pouvons nous poser la question de savoir à quoi servirait une base de données de documentation si personne n'y accède ?

Les utilisateurs de cette interface sont en fait de deux catégories : d'une part, les chercheurs, utilisateurs finaux par excellence, qui ont besoin d'accéder à Docdb pour y trouver les renseignements nécessaires à leur travail d'interprétation; d'autre part, les programmeurs qui utiliseraient cette interface pour vérifier l'état d'avancement de leur travail de construction de Docdb et comme documentation dans la réalisation de nouvelles bases de données.

L'accès à l'information peut se faire de trois manières différentes : premièrement, le progiciel SIR contient son propre module d'accès à la base de données, appelé FORMS; deuxièmement, SIR possède également un module SQL qui permet d'accéder aux données via des commandes SQL standards; une troisième possibilité enfin est d'écrire un programme d'interface dans le langage procédural du package, PQL.

La plupart des chercheurs étant des sociologues sans aucune expérience de l'informatique, la solution qui consisterait à leur demander de ne consulter Docdb que via SQL paraît peu réaliste. Il est en effet difficile d'imaginer que les chercheurs acceptent volontiers d'apprendre une méthode et un langage informatique assez ardu. La tentation chez eux serait alors de ne pas utiliser Docdb plutôt que de faire l'effort d'apprendre à réaliser ces requêtes SQL. De plus, une bonne connaissance de la structure de Docdb est nécessaire pour écrire ces requêtes. Cette connaissance étant déjà difficile pour les programmeurs (voir point 2.3), il paraît impossible de demander aux chercheurs d'utiliser SQL.

Pour ce qui est des programmeurs, le langage SQL n'est théoriquement pas un problème pour eux. Il nous faut cependant remarquer que le module SQL de SIR n'est pas d'une robustesse extrême : en essayant de comprendre la structure de Docdb par ce moyen, nous avons trouvé plusieurs requêtes tout à fait réalistes qui "plantaient" inexplicablement le système.

Une fois l'hypothèse SQL écartée, il nous reste à choisir entre réaliser une interface avec le module FORMS ou dans le langage PQL. Nous avons choisi de la développer en PQL car FORMS construit des interfaces trop élémentaires et peu

ergonomiques. De plus, FORMS est utile pour accéder de façon directe à une base de données, surtout pour y écrire, mais ne permet pas d'obtenir des champs provenant de plusieurs tables sur un même écran, ce qui rend la consultation très pénible et notre travail impossible. En effet, la plupart des renseignements demandés par les chercheurs s'expriment sous la forme de jointures de tables.

2.1.2 Pourquoi un prototype ?

Une fois acquis le besoin d'une interface et la nécessité de la réaliser nous-mêmes, se pose la question de savoir s'il faut écrire directement un programme complet ou s'il est préférable de réaliser d'abord un prototype. La réponse à ce dilemme a été influencée par deux considérations : d'une part, le manque d'expérience du CES dans le domaine de l'interface et, d'autre part, la difficulté de définir les besoins d'utilisateur non informaticiens.

Aucune gestion ou manipulation d'écrans en langage PQL n'ayant jamais été conçue au CES, il était assez difficile d'estimer ce qui était réalisable et le temps nécessaire pour le réaliser.

Le prototypage était alors d'abord utile en tant que laboratoire de recherche : le manuel de PQL donnait une description assez peu détaillée des fonctions de gestion d'écrans et il était donc nécessaire de les essayer au sein d'un prototype pour déterminer leurs possibilités et leurs limites.

De plus, il était difficile d'évaluer le temps nécessaire pour réaliser une application complète sans nous être au préalable exercés à la programmation d'interface en PQL. L'écriture d'un prototype nous apportait à cet égard des renseignements utiles.

Pour avoir une interface qui corresponde bien aux désidérata de l'utilisateur, il est important de définir le plus complètement possible ses besoins. Pour ce faire, un système incomplet peut être réalisé et présenté aux utilisateurs qui préciseront à ce moment leurs besoins en testant le prototype.

Les avantages de l'utilisation d'un prototype pour l'analyse des besoins sont, selon Sommerville :

- "les incompréhensions entre ceux qui développent le logiciel et les utilisateurs peuvent être identifiées au fur et à mesure que les fonctions du système sont présentées;
- les services demandés par l'utilisateur et oubliés peuvent être découverts;
- les services offerts à l'utilisateur difficiles ou confus peuvent être identifiés et raffinés;
- l'équipe qui réalise le logiciel peut trouver des besoins incomplets ou incohérents en développant le prototype;
- un système limité mais fonctionnant est rapidement disponible pour montrer au management la faisabilité et l'utilité de l'application;
- le prototype sert de base à l'écriture des spécifications pour un système final de qualité"[2].

Deux motifs complémentaires à notre choix de réaliser un prototype était le manque de temps et l'instabilité de Docdb (voir à ce sujet le point 2.3). En effet, il nous paraissait impossible de réaliser une application définitive en trois mois (en plus de comprendre le fonctionnement et les réalisations du CES et de son unité informatique). De plus, cette instabilité rendait également difficile le développement d'une application complète, capable d'évoluer en même temps que la base de données qu'elle doit consulter.

En accord avec Mme Lamb, nous avons donc décidé de travailler au développement d'un prototype dont autant de fonctionnalités que possible seraient réelles au lieu d'apparentes. Nous avons délibérement négligé la modularité et le caractère pluri-utilisateur que devrait posséder le programme final pour nous concentrer sur une interface ergonomique et rencontrant les besoins des chercheurs.

2.2 Détermination des besoins

Nous avons rencontré les chercheurs pour leur demander quels étaient leurs besoins. Nous avons ensuite déterminé ce qui était réalisable ou presque et ce qui ne l'était pas. Enfin, nous en avons déduit une synthèse des besoins des chercheurs.

---

[2] [Sommerville89], pg 113-114

2.2.1 Rencontre avec les chercheurs

Parmi les 7 chercheurs du CES, certains sont tout à fait familiers à l'utilisation de l'informatique et ne la craigne pas alors que d'autres sont réticents à l'usage direct de l'informatique, c'est-à-dire qu'ils utilisent des listings commandés à l'unité informatique pour leurs travaux de recherche mais ne souhaitent pas générer eux-mêmes ces listings.

Pour faciliter le contact et nous permettre de percevoir correctement les besoins des chercheurs (même des plus réticents), nous avons décidés de rencontrer chacun individuellement dans son bureau. Au cours d'une entrevue d'une durée de vingt à trente minutes, nous avons discuté librement de la façon dont ils voyaient l'accès idéal à Docdb. Nous avons noté toutes leurs idées, même celles qui paraissaient irréalisables.

La discussion s'est faite à plusieurs niveaux : nous avons essayé de savoir quelle utilisation ils feraient de l'application, quelles fonctionnalités leur seraient nécessaires, quel type d'interface ils désiraient et comment ils voyaient le déroulement d'une session.

2.2.2 Compilation des résultats et désagrégation

Nous avons ensuite rassemblé les demandes de chacun. Nous avons constitué, d'une part, un ensemble de considérations ergonomiques et, d'autre part, une grande liste des fonctionnalités requises.

Les demandes des utilisateurs au niveau ergonomique sont :

- accès en parallèle aux fichiers SPSS de texte;
- temps d'accès aussi réduit que celui nécessaire à l'accès à la documentation papier;
- pas de SQL;
- dialogue intuitif si possible sur base de menus (menu driven) comme certaines applications du CES;
- ils souhaiteraient la possibilité d'éviter (bypass) certains menus quand la phase d'apprentissage est terminée;
- une introduction pour les nouveaux venus et les utilisateurs extérieurs (et peut-être des exemples de fichiers SPSS) seraient utiles;

- un schéma des différents balayages (sweeps) d'une enquête devrait être disponible;
- avertissements et explications en cas d'erreur;
- possibilité de diriger un résultat vers l'imprimante.

Ces demandes d'ergonomie ont été prises en considération pour le fonctionnement de l'interface utilisateur (voir chapitre 3).

Après avoir éliminé les nombreuses redondances (la plupart des consultations demandées se retrouvait chez tous les chercheurs), nous avons réparti la liste des fonctionnalités requises en trois sous-listes : une liste des fonctions réalisables dans l'état actuel de Docdb, une liste des fonctions réalisables au prix de quelques aménagements à Docdb et une liste de fonctions impossibles à réaliser dans l'état actuel du support logique et physique de l'informatique au CES.

Les fonctions réalisables directement peuvent être résumées par les demandes suivantes :

- retrouver les informations sur une variable, un ensemble de données (dataset), une étude (study) ou une enquête (survey), étant donné son nom;
- si la même question apparait dans plusieurs enquêtes, donner les noms des variables qui y correspondent;
- dire où et comment une base de données ou un dataset est accessible. Donner la localisation, le type de dataset, la disponibilité, etc ...;
- donner les variables liées à un sujet (topic) particulier dans une ou plusieurs enquêtes;
- dire quelles sont les questions liées à un sujet particulier. Donner les questions et l'année (et éventuellement le dataset);
- dire quel(s) filtre(s)[3] influence(nt) une variable donnée. Retrouver la suite de filtres qui conduit à cette variable;
- étant donné un nom de variable, trouver l'enquête (et/ou le dataset) dans laquelle il apparaît. Donner le nom de fichier et la localisation physique;
- retrouver le texte de la question à travers les années pour un sujet ou un nom de variable donné;
- étant donné l'identifiant d'une enquête longitudinale (cohort survey), dire quelles sont ses variables;

---

[3] Par filtre, on entend une variable représentant le résultat d'une question filtre. Une telle variable détermine quelles questions suivent (elle peut également être utilisée pour des recoupements).

- donner la position d'une variable sur la page de questionnaire et sa place par rapport aux questions filtres;
- donner les valeurs manquantes[4] (missing values) pour une variable donnée;
- donner les fréquences d'une variable donnée et dire si elle est filtrée;
- dire quels datasets et études font partie d'une enquête donnée;
- étant donné un nom de variable, retrouver dans quel record elle se trouve.

Les fonctions réalisables au prix de quelques améliorations à Docdb sont :

- donner les différents textes d'une question, étant donné le nom de sa variable ou une variable filtrée et sa valeur;
- dire à qui une version donnée de la question a été posée;
- donner le contenu de la question sur le dos de la couverture du questionnaire (backpage prompt question) par an (si possible);
- donner les codes d'occupation et leurs variations au cours des ans. Si possible, expliquer les catégories conçues;
- donner les commentaires produits par Questmast (voir point 1.2).

Les fonctions demandées mais actuellement irréalisables sont :

- accéder par le terminal aux questionnaires, annotés ou non (c'est-à-dire disposer sur le terminal des pages du questionnaire, scannées);
- donner le contexte d'une enquête;
- étant donné le nom d'une variable, donner le codage nécessaire à l'inclure dans un ensemble de données;
- écrire automatiquement des procédures de traduction pour recoder;
- vérifier la cohérence avec la réalité (par exemple, si le métier est avocat, les études doivent être universitaires);
- générer automatiquement les règles pour faire coïncider ou joindre des datasets et/ou des variables (avec des messages d'avertissement sur les spécifications).

2.2.3 Synthèse des besoins réalisables

Nous avons ensuite séparé les requêtes réalisables en trois groupes : les informations sur les données principales, les recherches de base et les recherches évoluées.

---

[4] Ces valeurs (identifiantes) sont utilisées selon les circonstances, si la personne qui a répondu au questionnaire a négligé cette réponse, a donné une réponse incohérente,...

Par information sur les données principales, nous entendons les compléments d'information que l'utilisateur peut demander sur une variable, un dataset, une enquête ou une étude (par exemple, étant donné le nom d'une variable, trouver ses valeurs minimum et maximum et le texte de la question dont elle est la réponse).

Par recherche de base, nous comprenons les requêtes les plus courantes : dans l'ensemble formé par les cinq entités principales de Docdb (la variable, le dataset, l'étude, l'enquête et le sujet), la recherche de base doit permettre à l'utilisateur de trouver, à partir des entités qu'il connaît, les autres entités qu'il cherche.

Les recherches évoluées sont en fait les requêtes plus spécifiques demandées par certains chercheurs en fonction de besoins précis. Il s'agit de :

- trouver le chemin de variables filtres qui conduit à une variable donnée;
- donner la liste des variables pour une année et un sujet donnés;
- donner la liste des variables et des noms et numéros de record pour un dataset dont l'utilisateur connaît le nom;
- trouver les fréquences d'une variable
- donner les positions d'une variable donnée dans le questionnaire.

Avec quelques modifications à Docdb, nous pouvons ajouter les requêtes suivantes :

- trouver la liste des variables liées à une variable donnée;
- donner les différents textes de la question d'une variable donnée;
- trouver la liste des variables correspondant à une question.

Nous verrons au point 2.4 comment ces besoins ont été confronté à la situation de Docdb et les conséquences que ces sous-groupes ont eu sur l'implémentation du prototype.

2.3 Analyse de la structure de Docdb

Dans ce point, nous commençons par expliquer quels problèmes nous a posé Docdb. Nous poursuivons par la présentation du schéma de la base de données. Ensuite, nous montrons en deux temps ce que nous avons utilisé pour rencontrer ces problèmes.

2.3.1 Le problème de Docdb

Comme nous l'avons déjà fait remarquer précédemment, Docdb n'est pas une base de données figée. Au contraire, elle est constamment modifiée pour coller le plus possible aux besoins réels du CES.

Au point 2.1.1, nous avons vu pourquoi une interface d'accès à cette base de données était nécessaire pour son exploitation optimale. Une des raisons que nous avions invoquée à ce moment concernait directement la structure même de Docdb.

Nous allons nous attacher ici à montrer pourquoi et comment se modifie cette structure. Nous envisagerons ensuite l'impact de ces modifications.

La question du pourquoi est relativement simple. Bien que fort avancée, Docdb n'en reste pas moins une base de données provisoire, avec tout ce que cela implique de modifications et d'évolution. Nous savons qu'à terme Docdb doit remplacer les dictionnaires papier, mais nous ne savons pas encore quels seront les besoins futurs des chercheurs. De plus, nous ne pouvons pas non plus envisager tous les avantages et développement que peut apporter un tel outil : peut-être sera-t-il possible de créer, d'ici quelques années, un dataset de travail en répertoriant les variables pertinentes dans Docdb, tout cela selon un processus automatisé. Nous pensons donc que tant que Docdb reste un outil en évolution, elle ne pourra être considérée comme stable.

Il convient cependant de noter que l'évolution actuelle de la base de données est due principalement à deux raisons : la correction d'erreurs et les modifications apportées soit pour répondre à de nouveaux besoins, soit pour raison d'efficacité.

Cette évolution continuelle à plusieurs conséquences auxquelles nous allons nous attacher :

- incohérence de la structure physique;
- incohérence des données (complétude);
- distance entre spécifications et implémentation.

L'incohérence au niveau de la structure de la base de données découle de cette évolution trop rapide. Notons cependant que ces erreurs sont peu nombreuses et peuvent découler soit de la conception initiale, soit des modifications faites au cours du temps.

La deuxième conséquence, l'incohérence au niveau des données de la base, est déjà plus gênante si nous désirons utiliser Docdb de manière courante comme outil de recherche. Cette incohérence peut se marquer, par exemple, par l'incomplétude des données. Les causes de ce second problème sont, à notre avis, les suivantes :

- les questionnaires évoluent au cours des ans. Leur documentation (c'est-à-dire les métadonnées relatives à ces questionnaires) va évidemment suivre cette évolution, créant certaines disparités;
- bien qu'automatisés au maximum, tous les problèmes de transfert des métadonnées vers Docdb ne sont pas résolus. Nous citerons comme exemple le problème des différents textes d'une question.
- De plus, comme la structure de la base est changeante, elle ne favorise en rien la stabilité des données. Certaines données, probablement correctes au départ, peuvent avoir été invalidées avec le temps et les changements.

La troisième conséquence de l'évolution de Docdb, est la distance croissante entre les spécifications initiales et l'implémentation actuelle. D'une part, certaines parties du schéma conceptuel n'ont pas été implémentées (car le besoin ne s'en faisait s'en doute pas sentir), et d'autre part, certaines tables ont été ajoutées au schéma de Docdb sans pour autant que les spécifications soient modifiées. Le schéma que nous présentons au point suivant est relativement différent du schéma se trouvant dans le document fourni au ESRC.

2.3.2 Le schéma de Docdb

Pour donner au lecteur une meilleure idée de la structure de DocDB, nous allons décrire le schéma de Docdb. Celui-ci sera exprimé dans le formalisme MAG (Modèle d'Accès Généralisé) de J.-L. Hainaut[5].

La démarche que nous avons suivie est la suivante. Nous avons tout d'abord décrit le schéma conforme à SQL. Nous estimons cette étape nécessaire dans la mesure où nous ne disposions que de la description des tables SQL comme seule source fiable[6].

---

[5] Hainaut86, pg 20 et suivantes.
[6] Le listing des tables SQL se trouve dans l'annexe 1.

Le lecteur peut trouver la description complète des items des articles des spécifications initiales en annexe 1[7].

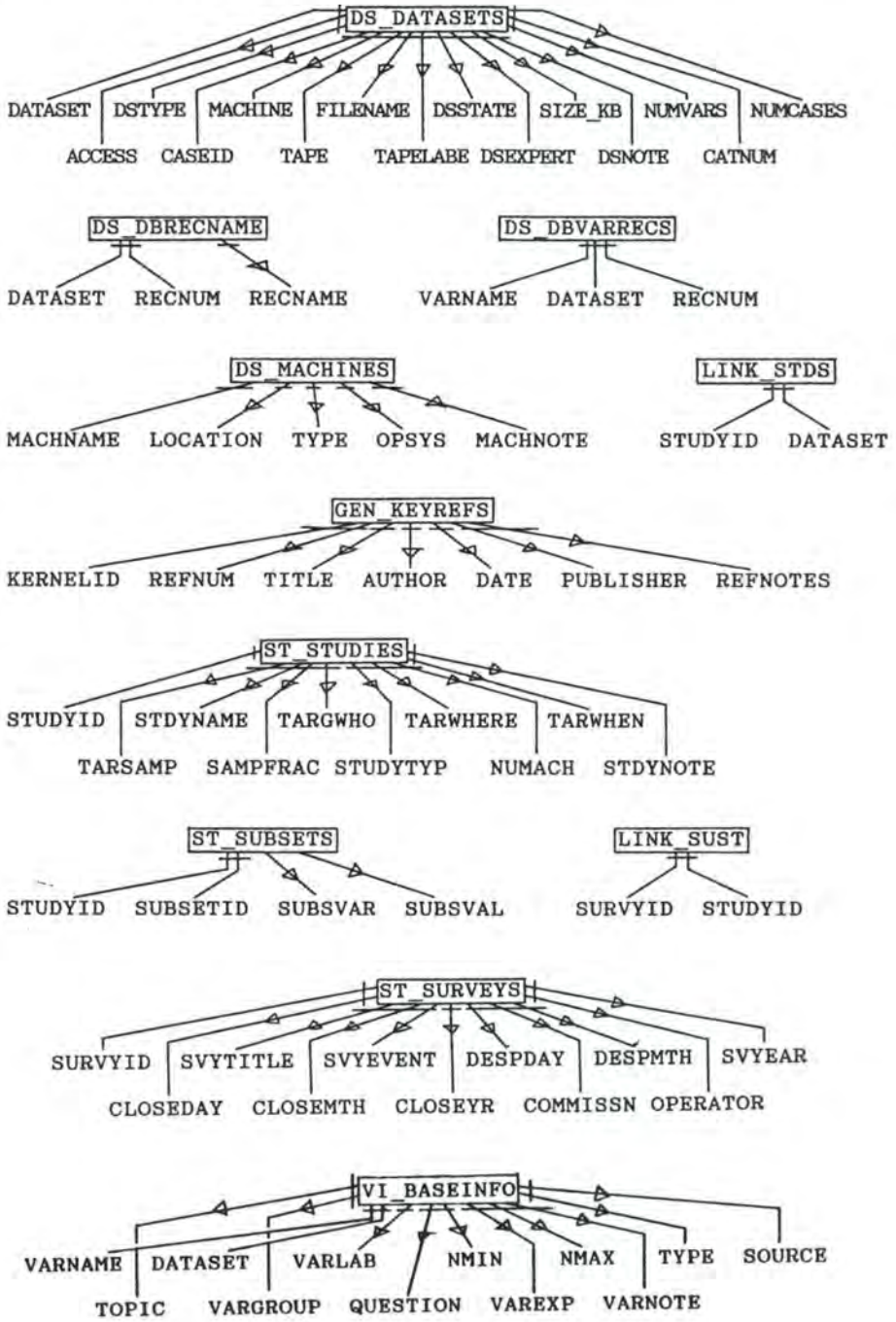


Figure 2.1 - Schéma conforme à SQL

[7] [Ritchie91], appendix 2, pg 2

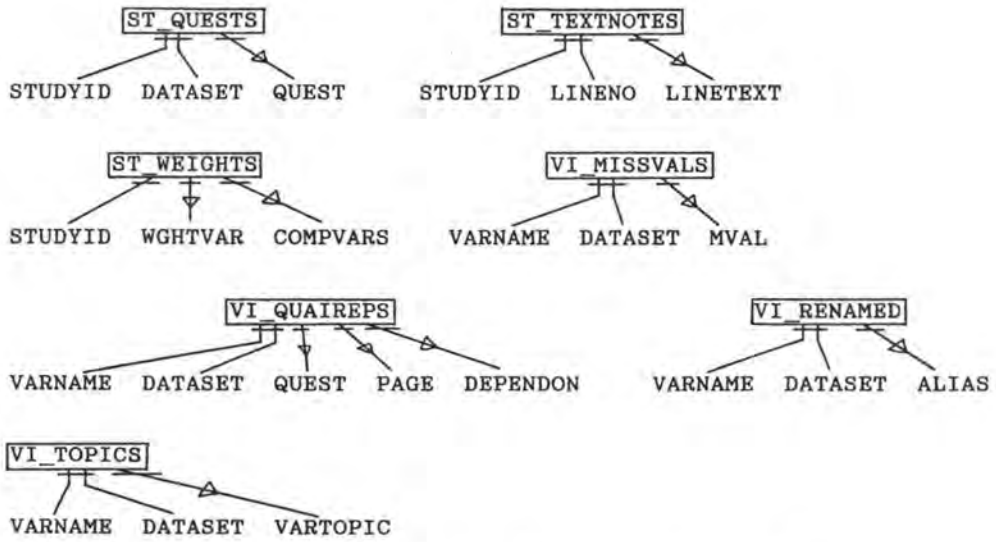


Figure 2.1 (suite) - Schéma conforme à SQL

A partir de ce premier schéma, dérivons maintenant le schéma MAG général.

Ce schéma décrit le support de recherche au CES. Nous n'exprimons dans le texte qui suit que les relations entre entités (articles). Nous ne nous attachons donc pas à décrire les items non identifiants.

Une enquête par questionnaire (su_survey) peut être utilisée dans plusieurs études (st_study). De même, une étude peut être basée sur plusieurs enquêtes. Chaque enquête ou étude est identifiée par son nom (surveyid et studyid). Une étude peut requérir l'usage d'un ou de plusieurs ensembles de données (ds_dataset). Bien entendu, un dataset peut être utilisé par différentes études.

Chaque dataset, identifié par son nom (dataset), est un fichier situé sur une machine (ds_machine). Nous supposons qu'un même dataset ne peut être distribué sur plusieurs machine, mais qu'une machine - également identifiée par son nom (machname)- peut contenir plusieurs datasets. Ceux-ci sont formés par un ensemble de variables (vi_baseinfo).

Chaque variable est identifiée par son nom (varname) et celui du dataset auquel elle appartient. Chaque variable appartient obligatoirement à un sujet (vi_topic). De ce fait, chaque sujet, identifié par son nom (vartopic) regroupe plusieurs variables.

Chaque variable possède une ou plusieurs valeurs correspondant aux valeurs manquantes (vi_missval).

machine(:Ds_datasets) in machname(:Ds_machine)
dataset(:Ds_dbrecname) in dataset(:Ds_datasets)
varname(:Ds_dbvarrecs) in varname(:Vi_baseinfo)
dataset(:Ds_dbvarrecs) in dataset(:Ds_datasets)
recnum(:Ds_dbvarrecs) in recnum(:Ds_dbrecname)
kernelid(:Gen_Keyrefs) in survyid(:Su_surveys)
kernelid(:Gen_Keyrefs) in studyid(:St_studies)
kernelid(:Gen_Keyrefs) in dataset(:Ds_datasets)
studyid(:Link_stds) in studyid(:St_studies)
dataset(:Link_stds) in dataset(:Ds_datasets)
survyid(:Link_sust) in survyid(:Su_surveys)
studyid(:Link_sust) in studyid(:St_studies)
studyid(:St_quests) in studyid(:St_studies)
dataset(:st_quests) in dataset(:Ds_datasets)
studyid(:St_subset) in studyid(:St_studies)
studyid(:St_textnotes) in studyid(:St_studies)
studyid(:St_weigths) in studyid(:St_studies)
dataset(:Vi_baseinfo) in dataset(:Ds_datasets)
varname(:Vi_derivhow) in varname(:Vi_baseinfo)
dataset(:Vi_derivhow) in dataset(:Ds_datasets)
varname(:Vi_filtervars) in varname(:Vi_baseinfo)
dataset(:Vi_filtervars) in dataset(:Ds_datasets)
varname(:Vi_labfreqs) in varname(:Vi_baseinfo)
dataset(:Vi_labfreqs) in dataset(:Ds_datasets)
varname(:Vi_missvals) in varname(:Vi_baseinfo)
dataset(:Vi_missvals) in dataset(:Ds_datasets)
varname(:Vi_quaireps) in varname(:Vi_baseinfo)
dataset(:Vi_quaireps) in dataset(:Ds_datasets)
varname(:Vi_renamed) in varname(:Vi_baseinfo)
dataset(:Vi_renamed) in dataset(:Ds_datasets)
varname(:Vi_topics) in varname(:Vi_baseinfo)
dataset(:Vi_topics) in dataset(:Ds_datasets)

Figure 2.2 - Contraintes supplémentaires pour le schéma conforme à SQL

Une variable peut être une variable filtre (vi_filtervar).

Pour chaque variable, nous disposons également d'informations concernant les etiquettes et leurs fréquences (vi_labfreq). La valeur correspondant au label est identifiante. De plus, si une variable est équivalente à une autre, nous disposons du nom du dataset et de la variable équivalente (vi_equivars). Une variable peut être équivalente à plus d'une variable.

Une autre information intéressante concerne le questionnaire dans lequel apparaît la variable (vi_quaireps). En fait; une variable peut apparaître dans plusieurs questionnaires. Un questionnaire est identifié par son nom (quest).

Une variable peut aussi avoir été renommée pour figurer dans un autre dataset (vi_renamed). La variable renommée est identifiée par son nom (alias) et son dataset. Une variable peut avoir été renommée plusieurs fois pour figurer dans différents datasets.

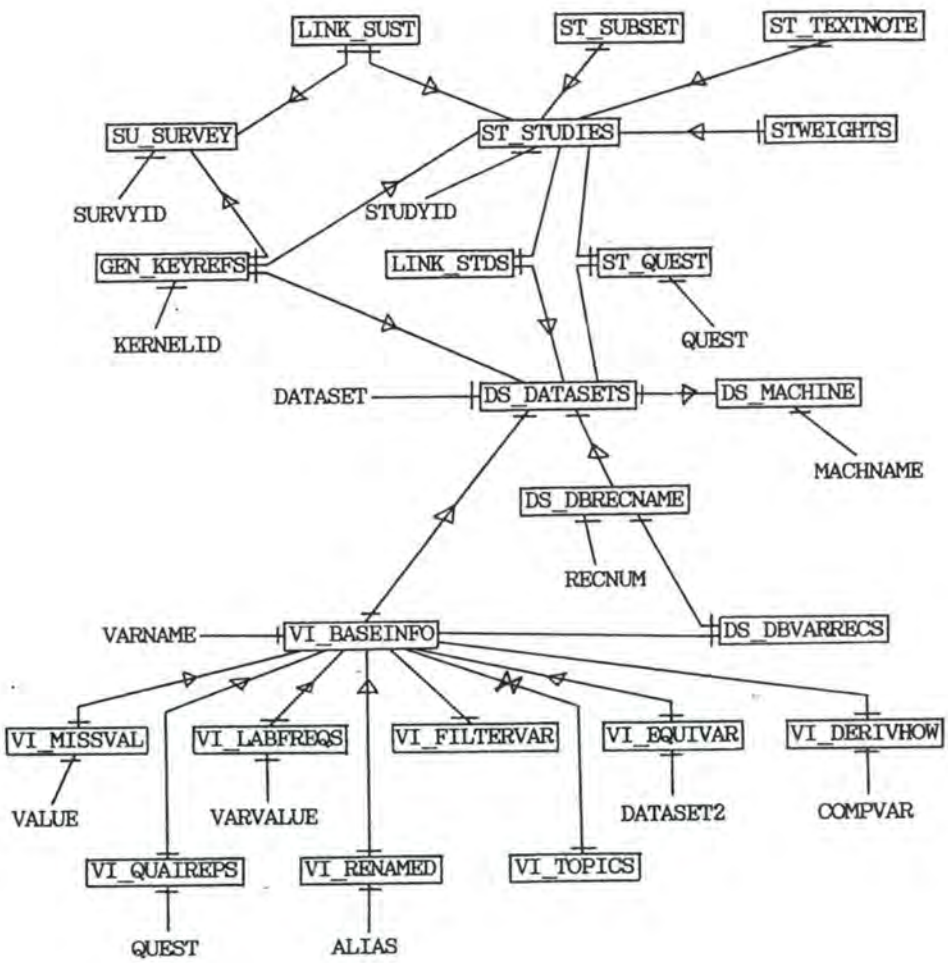


Figure 2.3 - Schéma des accès possibles de Docdb

Si une variable n'est pas liée à une question, cela signifie que soit elle provient d'une autre source, soit elle est dérivée (calculée) à partir d'autres variables. Dans le

second cas, nous devons disposer du nom des variables utilisées pour le calcul ainsi que du code déterminant ce calcul (vi_derivhow)

Finalement, nous obtenons le schéma des accès possibles présenté à la figure 2.3. Rappelons qu'il est indépendant de toute implémentation.

Nous avons constaté les deux problèmes suivants.

Certaines tables font partie du schéma mais ne contiennent aucune donnée[8]. Cela fait partie du problème de complétude dont nous avons déjà parlé. Par exemple, la table gen_keyrefs ou encore st_quests.

L'autre problème est une erreur de modélisation qui a été découverte lorsque les programmeurs ont voulu retrouver des données avec notre prototype. En effet, certains datasets sont des sous-ensembles d'autres datasets. Même si les données de documentation sont identiques, la structure actuelle de Docdb nécessite qu'elles soient répétées. Une relation récursive (parent-enfants) permettrait d'éviter cette redondance et améliorerait la cohérence des données.

2.3.3 Le "Constraint Checker"

Comme nous désirions que le prototype d'interface soit le plus fonctionnel possible, nous avons dû réduire au maximum l'étendue des problèmes que nous avons exposés au point 2.3.1.

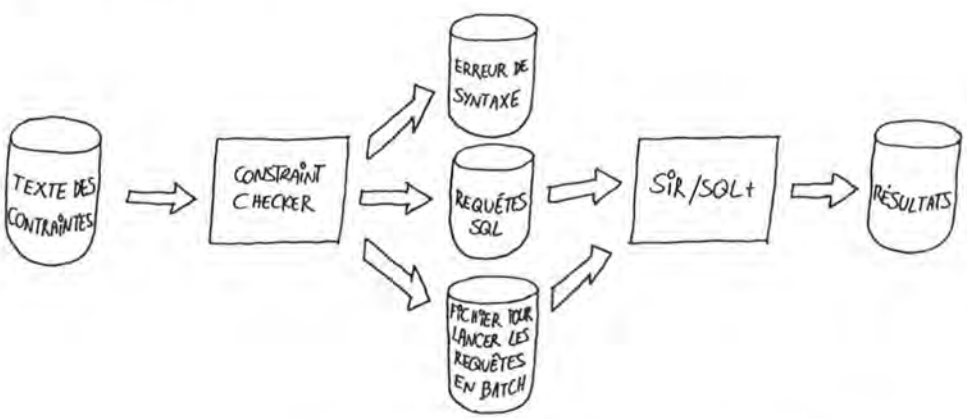


Figure 2.4 - Fonctionnement du Constraint Checker

[8] Le lecteur peut se référer à l'annexe 1 pour le remplissage de la base de données.

Les schémas MAG sont un premier pas dans cette direction puisqu'ils nous permettent de nous rendre compte des éventuelles erreurs de modélisations. Le constraint checker est un deuxième pas vers notre but, dans la mesure où il nous permet de vérifier la complétude et l'intégrité des données, du moins dans une certaine mesure.

L'outil que nous avons réalisé se présente sous la forme d'un programme PQL. Notons qu'il ne fait pas partie de notre prototype d'interface, ce n'est qu'un instrument de mise au point de la base de données.

En simplifiant, on peut décrire le fonctionnement de la manière suivante[9].

Le constraint checker utilise en entrée un fichier texte contenant les contraintes à vérifier. Il génère alors un autre fichier texte, contenant un ensemble de requêtes SQL. Chaque contrainte correspond à une requête. Il suffit alors d'exécuter l'ensemble des requêtes SQL pour la base de données adéquate et de consulter le fichier résultat qui est automatiquement généré.

Si le fichier résultat est vide, cela signifie que les contraintes sont vérifiées. A l'inverse, s'il s'avère qu'une contrainte n'est pas vérifiée, la requête fournira :

- le nom des tables SQL incriminées;
- les lignes où une incohérence apparaît.

Le fichier des contraintes possède une structure particulière. Il doit obligatoirement se terminer par une ligne contenant le mot "finish". De plus, il ne peut être formé que de lignes représentant les contraintes sous une des deux formes suivantes :

1. unique <nom de champ> in <nom de table>

2. <nom de champ>(:<nom de table>) in <nom de champ>(:<nom de table>) [and <nom de champ>(:<nom de table>) in <nom de champ>(:<nom de table>)]

Le fichier présenté à la figure 2.5 est un exemple correct de fichier pour le Constraint Checker. Il vérifie que les noms de datasets sont bien identifiants. Ensuite, il s'assure que tous les datasets repris dans la table vi_baseinfo sont bien des datasets de

[9] le manuel se trouve en annexe 2.

la table ds_dataset. Enfin, il vérifie que chaque variable (identifiée, rappelons-le, par leur nom et leur dataset) se trouvant dans la table vi_topic, appartient bien à la liste des variables de vi_baseinfo. Le fichier contient donc trois contraintes sur le schéma de Docdb.

```
unique dataset in ds_dataset

vi_baseinfo(:dataset) in dataset(:ds_dataset)
vi_topics(:varname) in vi_baseinfo(:varname) and
vi_topics(:dataset) in vi_baseinfo(:dataset)
finish
```

Figure 2.5 - Exemple d'utilisation du Constraint checker

Notons que la notation utilisée pour décrire les contraintes est librement inspirée de celle vue au cours par J.-L. Hainaut[10].

2.3.4 Propositions de modifications

Certaines modifications ont déjà été évoquées au point 2.2.2 en ce qui concerne les textes de questions.

Nous avons encore fait une proposition visant à améliorer les performances et à simplifier les requêtes PQL. Cette modification consiste à rajouter une table entre les tables su_survey et ds_datasets. Cette table, que nous appelons bridge, contiendrait les identifiants de chacune des deux tables qu'elle relie. Elle permettrait de déterminer, pour chaque dataset, les enquêtes dont il est issus, et pour chaque enquête, les datasets qui en dépendent. En annexe se trouvent les différentes requêtes pour construire la table et son index; de même que le programme PQL utilisé pour remplir la table[11].

Nous présentons ci-dessous le schéma MAG modifié.

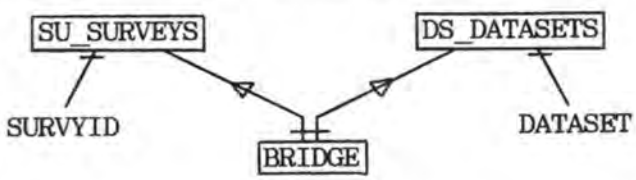


Figure 2.6 - La table bridge dans le schéma de Docdb

[10] [Hainaut86]
[11] voir annexe 3.

2.4 Architecture du prototype

Nous avons déterminé quels sont les besoins et étudié la structure de la base de données Docdb. Nous allons maintenant utiliser ces éléments pour déterminer l'architecture de notre programme.

2.4.1 Démarche

Dans un souci de facilité de lecture et de modifications, nous avons construit notre prototype de façon modulaire : nous avons par exemple regroupé dans une même famille du programme[12] la gestion des erreurs, de l'aide ou l'accès à la base de données.

La découpe de notre architecture a été basée sur deux considérations. D'une part, les fonctionnalités demandées par les chercheurs et, d'autre part, leurs désirs au niveau ergonomique.

2.4.2 Des fonctionnalités requises vers l'architecture

Un des besoins clairement défini est celui de pouvoir utiliser l'interface à Docdb en parallèle avec la consultation de texte SPSS qui sert de base au travail du chercheur. Une des grandes option du programme est donc la possibilité de visualiser un fichier texte. Notons qu'il n'est cependant pas possible de modifier celui-ci. Cette option doit évidemment disposer de plusieurs commandes classiques dans la consultation de texte : la possibilité de choisir et de changer de fichier à éditer, le saut vers une autre ligne, ... Nous avons appelé ce module "option Textfile".

Les trois groupes de besoins exprimés au point 2.2.3 peuvent aussi être regroupés chacun dans une famille.

Les informations sur les données principales doivent être disponibles à n'importe quel moment, y compris dans l'utilisation des recherches de base ou des recherches évoluées. Devant l'abondance d'informations pertinentes pour le dataset et la variable, nous avons décomposé ces informations en deux options différentes, "Basic" et "Advanced". Notons donc qu'il n'y a pas d'information "Advanced" pour les

---

[12] Rappelons qu'il n'y a pas de réel appel de procédure en PQL (voir point 1.4), mais que nous disposons d'une facilité de regroupement, les familles.

études et les enquêtes. Nous désignons globalement ces deux options par le terme "information".

Les recherches de base forment également un module à part entière. Nous avons vu dans l'analyse de la structure de Docdb que les cinq entités principales aux yeux des chercheurs sont parmis les tables les plus accessibles de Docdb. Etant donné qu'un certain nombre (y compris zéro) d'entités de base sont connues, cette option, que nous avons appelé "List", fournit une liste d'entités parmis les entités inconnues dont les valeurs sont connexes aux valeurs d'entrées. Notre problème a été de déterminer pour toutes les combinaisons d'entrée possibles quelles seraient les combinaisons de sortie les plus pertinentes.

Pour terminer, nous avons regroupé les recherches évoluées sous le vocable "Others". Les différentes requêtes spécifiques seront choisies au sein de cette option via un menu.

2.4.3 De l'ergonomie souhaitée vers l'architecture

En nous basant sur les demandes des chercheurs en matière d'ergonomie, nous avons déterminé quatre modules supplémentaires : un setup, une option de mémorisation d'écran, un module d'aide et une gestion d'erreur centralisée.

La demande de souplesse d'utilisation, c'est-à-dire un certain paramétrage, et le souhait de pouvoir éviter certains menus lorsque la phase d'apprentissage est terminée, nous ont conduit à réaliser une option, nommée "Setup", appellable à n'importe quel moment de la session, dans laquelle l'utilisateur peut adapter à sa convenance divers paramètres.

Rappelons qu'il existe en PQL la possibilité de mémoriser un écran et de le restituer à volonté. Cette notion de template nous a fourni le moyen d'aider l'utilisateur en soulageant sa mémoire à court terme. Cette option, que nous avons appellée "Screen", lui permet par exemple d'y ranger une page du texte sur laquelle il travaille ou une page d'information qu'il a consultée précédemment.

C'est également un souci d'ergonomie qui nous a conduit à offrir une aide contextuelle. Il est possible à chaque étape du travail d'accéder à une ou plusieurs pages d'explications en relation avec la tâche en cours. Notons que ce module peut également rencontré deux autres besoins des chercheurs, à savoir fournir une

introduction et un schéma des différents balayages d'une enquête aux nouveaux venus et aux utilisateurs extérieurs.

Une dernière demande des chercheurs était d'être informés de leurs erreurs. Dans un souci de modularité, nous avons regroupé toute la gestion d'erreurs à un seul endroit.

Notons que la demande de pouvoir imprimer des résultats n'a pas été prise en compte par manque de temps.

2.4.4 Architecture globale

La figure ci-dessous résume la façon dont s'articulent les différents modules de notre prototype.

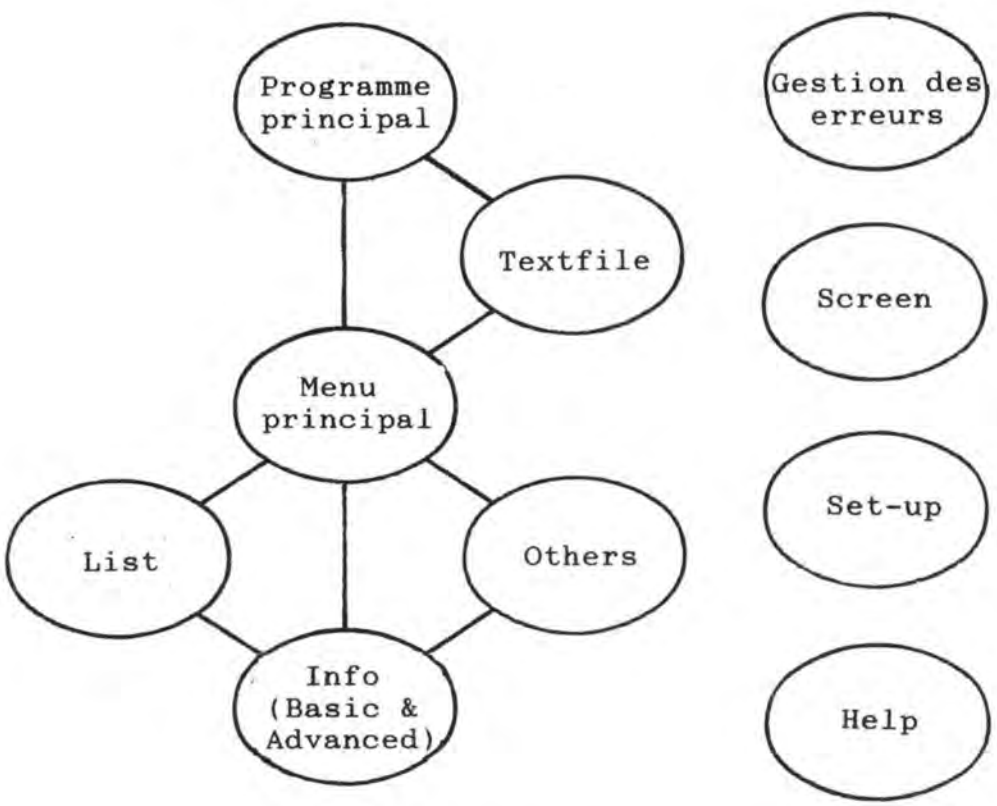


Figure 2.7 - Architecture globale

Notons que les quatre modules issus des besoins d'ergonomie sont appelables à partir de n'importe quels autres.

2.5 Spécification des accès à Docdb

2.5.1 Méthodologie

Notre manière de travailler pour implémenter les fonctions d'accès sur la base de données a été d'utiliser Docdb telle qu'elle était. La seule modification que nous nous soyons permis de faire, est l'adjonction de la table bridge pour les raisons citées plus haut (voir point 2.3.4).

En ce qui concerne la programmation, le grand principe que nous avons suivi est, lorsqu'un choix se présente, de placer les boucles nécessitant le moins d'itérations le plus à l'extérieur possible. Cela n'est bien entendu réalisable que pour certains cas, lorsque la condition de boucle est évaluable. Avec le placement des index, c'est le seul moyen que nous avons réellement pour agir sur la vitesse d'exécution des requêtes.

Le résultat des requêtes est placé dans une base de données, nommée INTERFC. La description complète du schéma de celle-ci se trouve en annexe. Ceci est rendu nécessaire par le fait que PQL n'est pas apte à gérer des variables dynamiques.

2.5.2 Classification des accès

Nous avons repris les trois grands groupes de fonctions vus au point 2.4, à savoir Info, List et Others. Le module INFO contient toutes les fonctions nécessaires lors de l'appel des options Basic et Advanced du prototype. Le module List, lui, contient toutes les fonctions nécessaires à l'utilisation de l'option List. La liste des combinaisons offertes est exhaustive. Le lecteur trouvera cette liste en annexe 4. Enfin, le module Others comprend les fonctions utilisées lorsqu'une des possibilités du menu Others est invoquée.

Nous présenterons, au point suivant, chacune des fonctions ainsi que leurs préconditions et postconditions.

2.5.3 Spécification des fonctions

Pour chaque groupe de fonctions, nous procédons de la manière suivante. Nous précisons tout d'abord où se retrouvent les résultats des requêtes. Ensuite, nous

donnons les préconditions et postconditions valables pour tout le groupe. Vient enfin la liste de toutes les fonctions, avec leur description.

Les fonctions du groupe LIST

La notation utilisée ici est la suivante : nous avons assigné une lettre à chacun des cinq champs possibles d'entrée (dataset - D, varname - V, study - Y, survey - S, topic - T) ainsi que la lettre E pour Empty. Chaque nom de fonction comprend deux groupes de lettres séparés par le chiffre "1". Le groupe de gauche représente les données en entrée, le groupe de droite les données en sortie. Par exemple, ST1VDY, signifie que l'utilisateur veut connaître les noms de variables, de datasets et d'études qui portent sur un sujet donné, repris dans une enquête donnée.

Le résultat de ces fonctions est écrit dans une table appellée Search de la base de données INTERFC.

Précondition :

- existence d'une table nommée Search dans la base de données INTERFC. La structure complète de la table est donnée en annexe.

Postconditions :

- - Le record zéro existe toujours après l'exécution de la fonction. Il contient soit la valeur de Errcode ou de Nline dans son premier champ. Si le nombre dans le premier champ est négatif, alors c'est le code d'erreur, sinon c'est le nombre de lignes. Les enregistrements utiles de la base de données sont numérotés de 1 à n.
- Si aucune erreur n'est rencontrée, la variable errcode est égale à zéro. Dans le cas contraire, elle contient le numéro de l'erreur.
- La variable nline est égale à zéro si une erreur est apparue. Sinon, elle contient le nombre de lignes utiles dans la table.

Les codes d'erreurs sont les suivants. Ils portent sur les données fournies en entrée :

1. le dataset n'existe pas dans DocDB;
2. la variable dont on a fournit le nom n'existe pas dans le dataset spécifié;
3. l'étude spécifiée n'est pas dans DocDB;
4. l'enquête donnée n'est pas dans DocDB;
5. le sujet spécifié n'existe pas dans la base de données

6. le dataset fourni ne fait pas partie de l'enquête spécifiée;
7. le dataset fourni ne fait pas partie de l'étude spécifiée;
8. l'étude n'est pas basée sur l'enquête spécifiée.
9. le nom de la variable n'existe pas dans DocDB
10. le sujet fourni n'existe pas pour les noms de variable et de dataset spécifiés.

Nous renvoyons le lecteur au tableau ci-dessus pour la liste des fonctions. Etant donné la structure de leur nom, leur effet est évident. Précisons simplement la fonction E1D : lorsque l'utilisateur choisit l'option List sans avoir rempli aucun des champs d'entrée, la liste des datasets de DocDB est produite.

Les fonctions du groupe OTHERS

Le résultat de ces fonctions est écrit dans une table appellée Menu de la base de données INTERFC.

Précondition :

- existence d'une table nommée Menu dans la base de données INTERFC. La structure complète de la table est donnée en annexe.

Postconditions :

- Les postconditions sont les mêmes que pour les fonctions du groupe List.

Les fonctions sont les suivantes.

DLREC : on veut obtenir la liste de tous les noms de records, leur numéro et les variables associées pour un dataset donné.

QLVAR : produit la liste de tous les noms de variables et de leur dataset liés à un questionnaire donné. Cette fonction n'a pas été implémentée car la structure actuelle de DocDB ne le permettait pas.

TLVYEAR : étant donné une année et un sujet, la fonction renvoie tous les noms de variables, ainsi que leur dataset, étude(s) et enquête(s).

VLEQUIV : étant donné un nom de variable et son dataset, la fonction fournit la liste de tous les noms de variables (et de leur dataset) qui sont équivalents.

VLPATH : étant donné une variable et son dataset, la fonction fournit le chemin des questions filtres qui y mène. La liste se présente sous la forme de paires de noms de variable et de dataset.

VLWORD : cette fonction produit la liste des textes de la question, ainsi que les raisons des différents wordings, étant donnés un nom de variable et son dataset. Cette fonction n'a pas été implémentée, DocDB ne disposant pas des informations.

Les fonctions du groupe INFO

Pour ce groupe, chaque fonction dipose de sa propre table pour enregistrer ses résultats. Cette distinction est due à la disparité des types d'informations que l'on récupère dans la base de données. Le nom de la table utilisée est le même que celui de la fonction.

Pour la signification des variables en entrée et en sortie, on se reportera à la description qui accompagne le schéma MAG compatible SQL.

Précondition

- la table nécessaire pour la fonction existe dans la base de données INTERFC.

Postcondition

- Les postconditions sont les mêmes que pour les fonctions des groupe List et Others.

D1INFO

Entrée : DATASET

Sortie : DSTYPE, MACHINE, DSSTATE, NUMVARS, NUMCASES, CASEID, DSEXPERT, DSNOTE

Cette fonction fournit les informations de bases concernant un dataset.

D2INFO

Entrée : DATASET

Sortie : FILENAME, SIZE_KB, ACCESS, TAPE, TAPELABE, CATNUM

Cette fonction fournit les informations "avancées" concernant un dataset.

S1INFO

Entrée : STUDYID

Sortie : STDYNAME, TARGWHO, TARWHERE, TARWHEN, TARSAMP, SAMPFRAC, STUDYTYP, NUMACH, STDYNOTE

Cette fonction fournit les informations concernant une étude.

U1INFO

Entrée : SURVYID

Sortie : SVYTITLE, SVYEVENT, DESPDAY, DESPMTH, SVYEAR, CLOSEDAY, CLOSEMTH, CLOSEYR, COMMISSN, OPERATOR

Cette fonction fournit les informations concernant une enquête.

V1FILTR

Entrée : VARNAME, DATASET

Sortie : TRUE/FALSE

Cette fonction renvoie vrai si la variable donnée en entrée est une variable filtre. La valeur VRAI est exprimée par le chiffre 1 en ligne 0 de la table V1FILTR.

VLFREQ

Entrée : VARNAME, DATASET

Sortie : list of (VARVALUE, VALLABEL, ABSFREQ, UWPCT)

Cette fonction produit la liste de tous les fréquences, pondérations, ... pour une variable donnée

V1INFO

Entrée : VARNAME, DATASET

Sortie : VARLAB, NMIN, NMAX, TYPE, SOURCE, VARGROUP, QUESTION, VAREXP, VARNOTE

Cette fonction fournit les informations de bases concernant une variable. Rappelons qu'une variable est identifiée par son nom et son dataset.

VLPOSIT

Entrée : VARNAME, DATASET

Sortie : List of (QUEST, PAGE, DEPENDON, PORDER)

Cette fonction fournit les informations relatives à la place de la variable dans son questionnaire.

VLREC

Entrée : VARNAME, DATASET

Sortie : list of (RECNUM, RECNAME)

Cette fonction fournit la liste de tous les records où l'on peut trouver la variable.

VLMISS

Entrée : VARNAME, DATASET

Sortie : List of (MVAL)

Cette fonction fournit la liste des valeurs manquantes possibles pour la variable.

Les fonctions de récupération

Chacune des fonctions présentées ci-dessus se voit associé à une autre fonction dite de "récupération". Cette dernière permet d'obtenir le contenu d'une ligne donnée de la table où sont rangés les résultats de la fonction "maître". Par exemple, la fonction Rvlmiss permet de récupérer les données fournies par VLMISS.

L'utilisateur d'une fonction de récupération doit préciser le numéro de la ligne à aller rechercher. Ce numéro est placé dans la variable globale LNUM avant l'appel. Les préconditions et postconditions sont les mêmes pour toutes les fonctions.

Préconditions

- la table n'est pas vide
- 0<= LNUM <= NLINE

Postcondition

- Les bonnes variables ont été mises à jour. Ces variables dépendent de la fonction sur laquelle se fait la requête.

Chapitre 3

Description du prototype

Dans ce chapitre, nous présenterons le fonctionnement global de notre prototype, tel que le perçoit l'utilisateur. Nous analyserons également son ergonomie pour en dégager les qualités et les défauts.

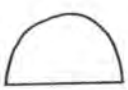
3.1 Comportement global

Après avoir montré l'enchaînement des écrans, nous détaillerons le comportement de chacun d'eux.

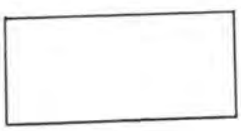
3.1.1 Enchaînement des écrans

Introduisons pour commencer les conventions que nous avons utilisées pour représenter graphiquement l'enchaînement des écrans.

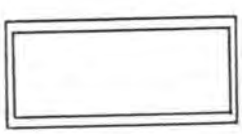
représente le début de la session.



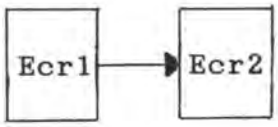
représente la fin de la session.



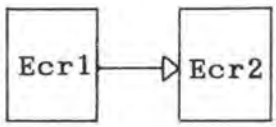
représente un écran. Celui-ci est soit un écran de menu, soit un écran d'information.



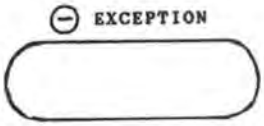
regroupe différents écrans pour lesquels les appels et les retours d'appel sont identiques.



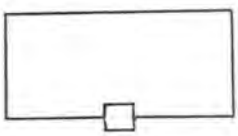
représente l'appel d'un écran Ecr2 par l'écran Ecr1.



représente le retour d'appel d'un écran Ecr1 vers un écran Ecr2.



représente un écran appelable de n'importe quel écran de l'application, sauf éventuellement de l'écran Exception.



Ce symbole, dessiné sur la ligne inférieure d'une boîte écran, signale que c'est en réalité une boîte de dialogue.

Nous pouvons maintenant exprimer l'enchaînement des écrans :

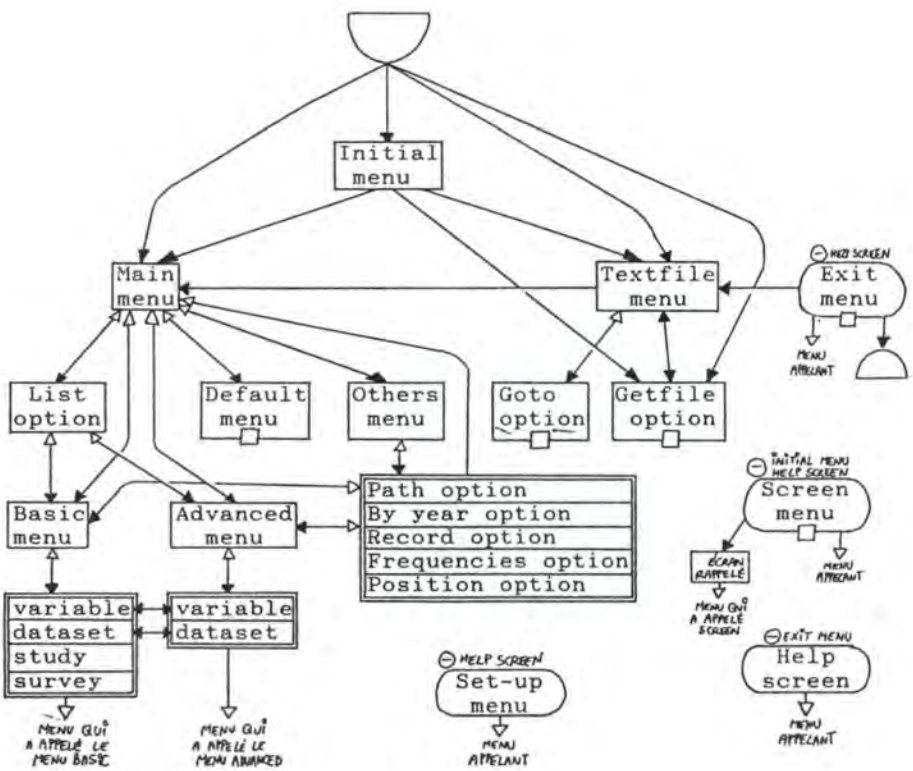
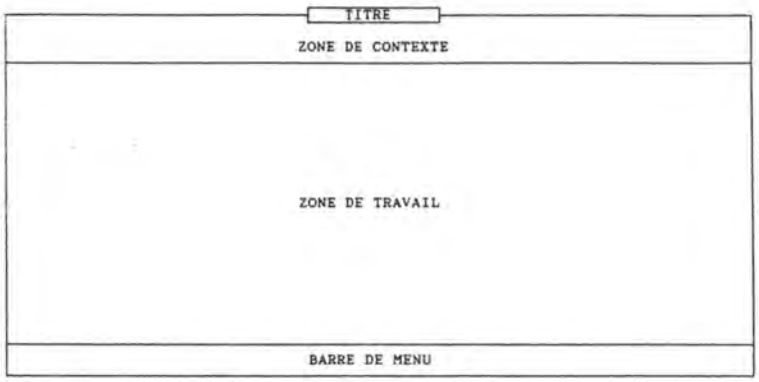


Fig 1. Enchaînement global des écrans

Avant de détailler chacun de ces écrans, précisons par un schéma les éléments constituants d'un écran type du prototype.

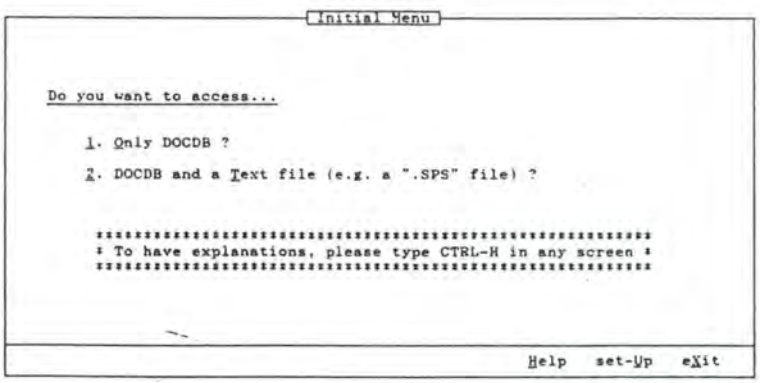


Nous allons utiliser ces différents éléments dans la description qui suit.

3.1.2 Le menu initial (Initial Menu)

Cet écran permet à l'utilisateur de choisir entre accéder uniquement à la base de données de documentation ou en parallèle à une visualisation d'un fichier texte.

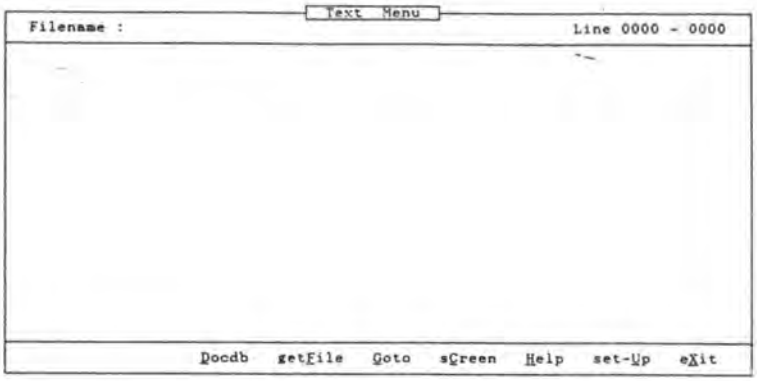
Le message entre étoiles dans la moitié inférieure de la zone de travail invite le nouvel utilisateur à appeler la fonction d'aide qui lui donne les explications nécessaires à la poursuite de la session. Signalons notamment que les options de la barre de menu sont accessibles par la combinaison de touche <CTRL> + <la lettre soulignée>.



Remarquons que grâce à l'option Set-up, il est possible d'éviter cet écran lors des utilisations ultérieures. Par exemple, un utilisateur confirmé qui ne travaille jamais avec un fichier texte préférera directement atteindre le menu principal sans passer par le menu initial. Si l'utilisateur désire néanmoins passer par ce menu, il peut également recourir à l'option de Set-up pour éliminer le message invitant à accéder à l'aide (message inutile pour un utilisateur chevronné).

3.1.3 L'option Texte

Cette option permet de visualiser un fichier texte. Le chercheur peut l'employer avec un fichier SPSS, qui est la base de son travail de recherche, alors que le programmeur peut choisir, par exemple, de parcourir ses programmes de remplissage de DOCDB tout en vérifiant la cohérence ou la complétude de son travail.

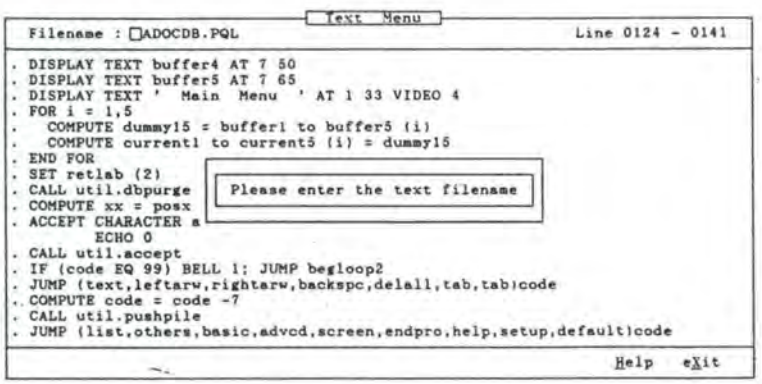


L'utilisateur peut parcourir le texte avec les flèches et les touches <Page up> et <Page down>. A n'importe quel moment, il peut basculer dans le menu principal et rechercher dans DOCDB toute information pertinente à son travail.

La zone de contexte contient deux informations: le nom du fichier courant et les numéros de lignes de texte affichées dans la zone de travail. La barre de menu offre notamment les options "Docdb", "getFile" et "Goto".

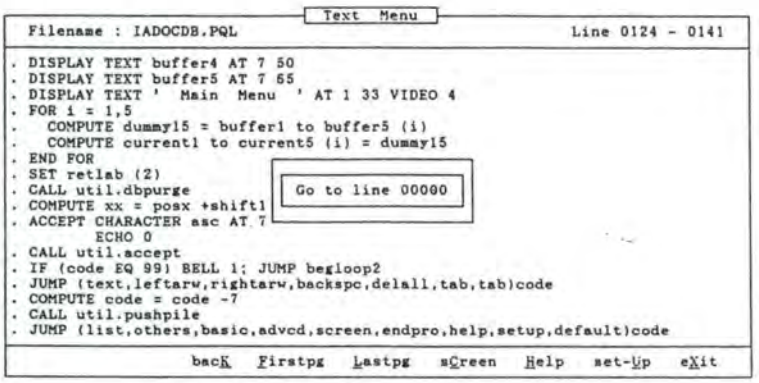
La première permet de passer au menu principal de l'interface à Docdb. Notons au passage que, dans le cas d'un retour au menu Texte, les variables du menu principal sont conservées jusqu'à l'appel suivant à Docdb.

Le choix de l'option getFile affiche, en surimpression au centre de la zone de travail, un message enjoignant l'utilisateur de spécifier un (nouveau) nom de fichier texte. Le nom est à entrer directement dans la zone de contexte.



Cette option est automatiquement proposée par l'application si aucun nom de fichier texte

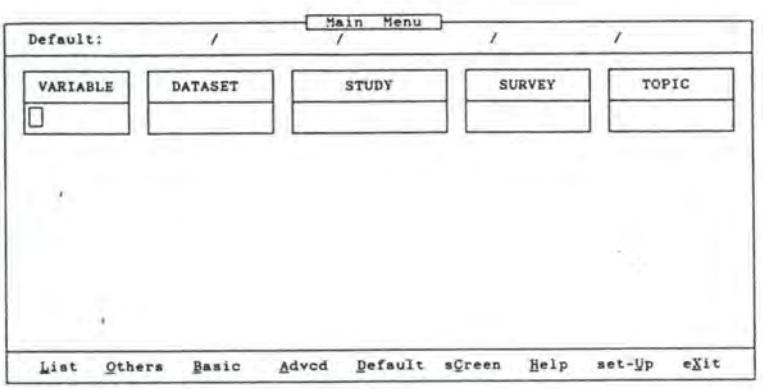
n'est défini. Il est possible, à l'aide du Set-up, de demander que, au début de la session, soit chargé le fichier texte utilisé à la fin de la précédente.



L'option Goto provoque l'apparition au milieu de la zone de travail d'une boîte de dialogue. L'utilisateur y est invité à donner le numéro de la ligne à laquelle il désire se rendre. La barre de menu de cette option comporte deux fonctions "FirstPg" et "LastPg" qui permettent respectivement de se rendre au début et à la fin du texte. Après la première utilisation de cette commande, le numéro de ligne du dernier appel est affiché par défaut (ce qui permet de sauter à une autre ligne puis de revenir là où on se trouvait).

3.1.4 Le menu principal (Main Menu)

Comme le lecteur peut le voir sur ce schéma, la zone de travail comporte cinq champs représentant chacun une entité de base issue de l'analyse de la structure. La zone de contexte contient éventuellement les valeurs mémorisées pour chacun de ces cinq champs, appelées "default values" (nous y reviendrons plus loin lorsque nous parlerons de l'option Default).



Pour commencer, nous pouvons introduire du texte dans chacun des champs. Pour ce faire, nous pouvons entrer les caractères dans le champ courant (celui où se trouve le curseur), utiliser les flèches pour s'y déplacer, utiliser <RETURN> ou <TAB> pour passer de l'un à l'autre ou bien effacer un caractère, un champ ou tous les champs (pour ce faire, l'utilisateur entre le caractère <DELALL>, voir option Set-up). Evidemment, il convient de ne remplir que les champs dont nous connaissons la valeur.

Une fois que cela est fait, nous pouvons choisir soit l'option "List" (pour que les champs vides se complètent), soit les options "Basic" ou "Advcd" (pour obtenir les informations sur les données principales déjà définies), soit enfin l'option "Others" (pour réaliser des requêtes évoluées à partir des champs connus).

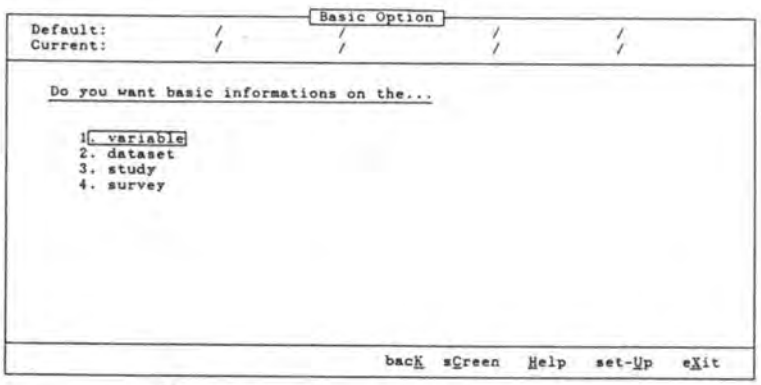
Nous allons maintenant détailler toutes les options de la barre de menu.

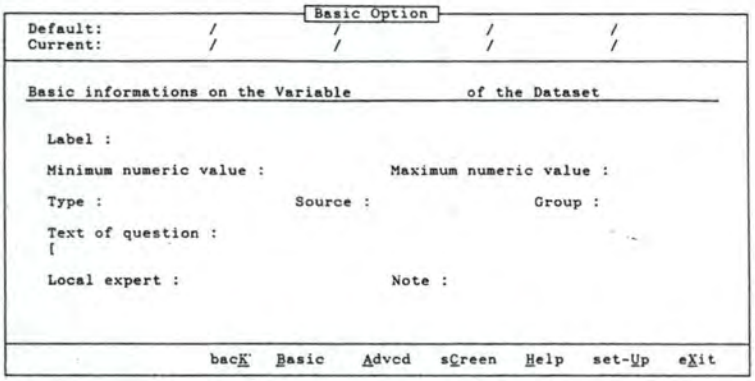
3.1.5 Les options d'information (Basic et Advanced Options)

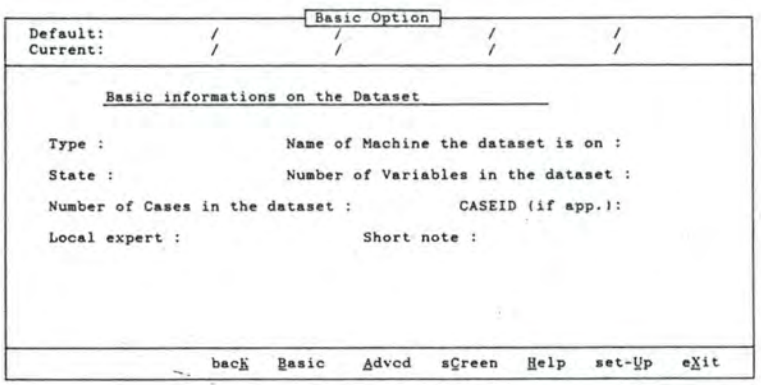
L'appel à une de ces deux options cause un élargissement de la zone de contexte. En plus des valeurs mémorisées, s'ajoutent les valeurs courantes des entités sur lesquelles se base cette recherche d'information. Ces valeurs courantes sont soit les valeurs des entités de base du menu principal (si l'appel y a lieu), soit les valeurs en contrasté sur les écrans, à partir desquelles l'option est appelée.

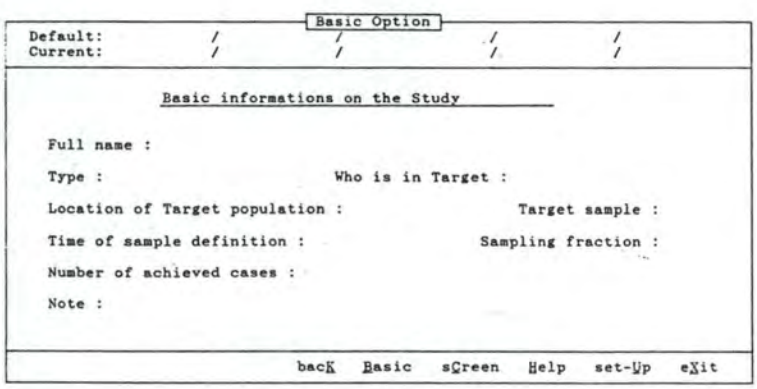
La zone de travail contient un menu qui est fonction des informations disponibles : si aucun champ n'est rempli dans le menu principal ou si les informations sont demandées à un moment où il n'y a aucune variable, aucun dataset, aucune étude ou enquête dans les champs courants, l'accès à cette option n'est pas permis. Si une partie seulement des champs est remplie, le menu ne présente que les choix disponibles; si un seul de ces champs est garni, l'utilisateur ne se voit pas présenter cet écran et se retrouve directement dans l'écran d'information correspondant. Rappelons que pour les études et les enquêtes, il n'y a pas d'informations avancées.

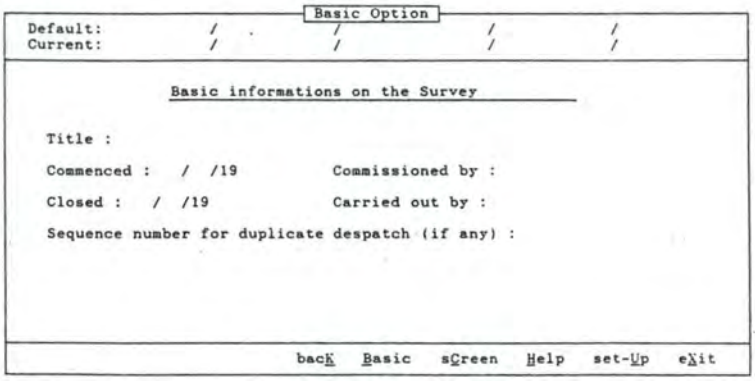
Voici les écrans d'information :

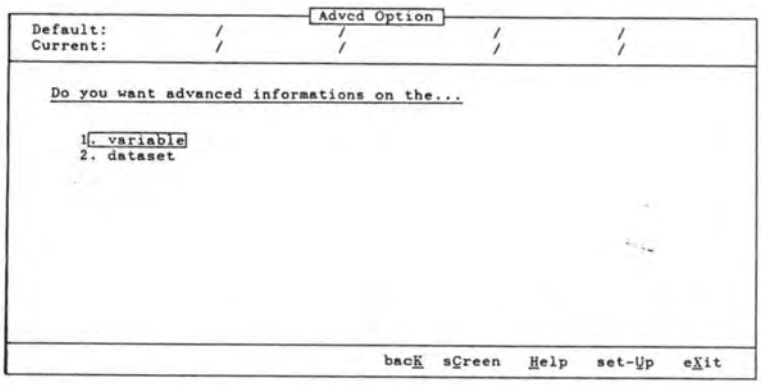


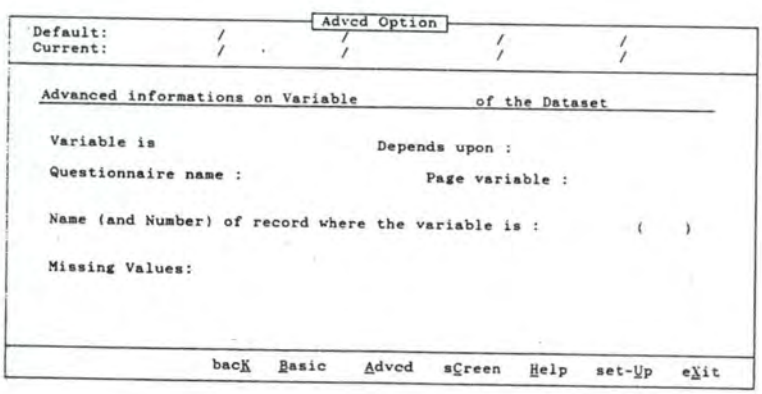

Basic Option
Default: / / / /
Current: / / / /
Basic informations on the Variable of the Dataset
Label :
Minimum numeric value :
Maximum numeric value :
Type :
Source :
Group :
Text of question :
Local expert :
Note :
bacK Basic Advcd sCreen Help set-Up eXit

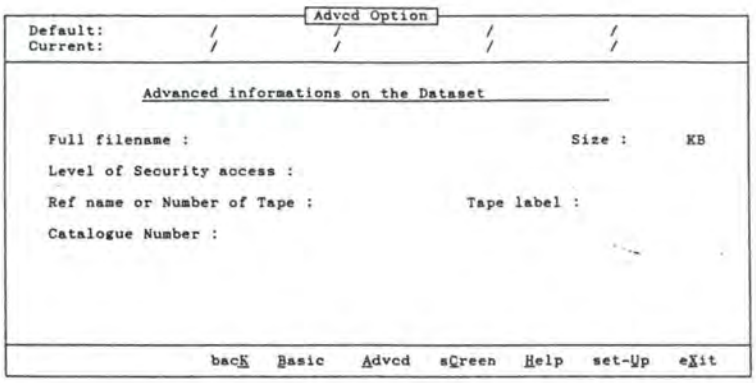

Basic Option
Default: / / / /
Current: / / / /
Basic informations on the Dataset
Type :
Name of Machine the dataset is on :
State :
Number of Variables in the dataset :
Number of Cases in the dataset :
CASEID (if app.):
Local expert :
Short note :
bacK Basic Advcd sCreen Help set-Up eXit


Basic Option
Default: / / / /
Current: / / / /
Basic informations on the Study
Full name :
Type :
Who is in Target :
Location of Target population :
Target sample :
Time of sample definition :
Sampling fraction :
Number of achieved cases :
Note :
bacK Basic sCreen Help set-Up eXit


Basic Option
Default: / . / / /
Current: / / / /
Basic informations on the Survey
Title :
Commenced : / /19
Commissioned by :
Closed : / /19
Carried out by :
Sequence number for duplicate despatch (if any) :
backK Basic sCreen Help set-Up eXit


Advcd Option
Default: / / / /
Current: / / / /
Do you want advanced informations on the...
1. variable
2. dataset
backK sCreen Help set-Up eXit


Advcd Option
Default: / / / /
Current: / / / /
Advanced informations on Variable of the Dataset
Variable is
Depends upon :
Questionnaire name :
Page variable :
Name (and Number) of record where the variable is : ( )
Missing Values:
backK Basic Advcd sCreen Help set-Up eXit

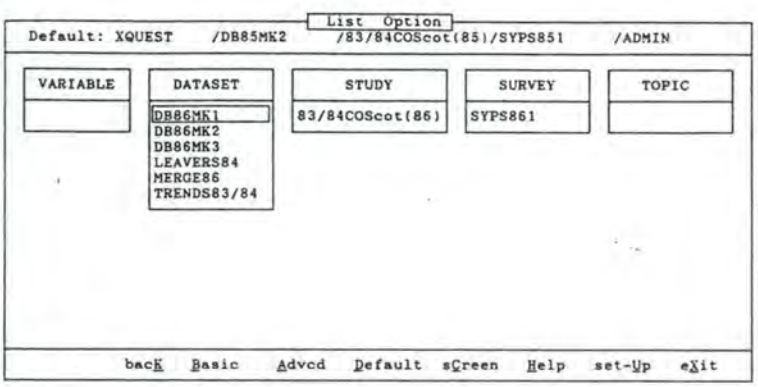


Lorsque l'utilisateur se trouve dans un des écrans ci-dessus, en plus des fonctions "sCreen", "Help", "setUp" et "eXit", il peut :

- retourner au menu par lequel il a choisi le champ sur lequel il désire des informations (Basic menu ou Advanced menu);
- retourner à l'écran appelant;
- accéder aux autres informations disponibles (uniquement pour la variable et le dataset), c'est-à-dire à Advanced si les informations de base sont affichées ou à Basic si ce sont les informations avancées.

3.1.6 L'option de liste (List Option)

Cette option peut être appelée du menu principal sauf si les cinq champs de celui-ci sont complétés. Ayant défini de zéro à quatre champs, l'utilisateur se voit présenter une liste de champs parmi les champs vides. Par exemple, si les champs Survey et Study sont complétés, l'option déroulera le champ Dataset avec une liste des datasets associés à ces enquêtes et études.



Rappelons que les combinaisons données et résultats se trouvent au point 2.5.4.

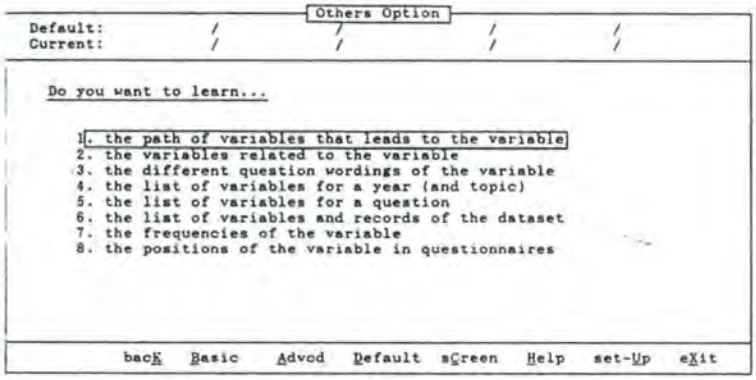
La ligne courante est en contrasté. L'utilisateur se déplace avec les flèches haut et bas de ligne en ligne et avec les flèches <PGUP> et <PGDOWN> de page en page. Notons que si la liste comporte plus d'une page, le signe "v" apparaît dans le bas de la zone de travail, à gauche du premier champ déroulé.

Si un choix convient à l'utilisateur, il lui suffit de frapper la touche <RETURN> pour se retrouver dans le menu principal avec les champs garnis des valeurs sélectionnées.

Remarquons qu'il est possible en parcourant la liste de faire appel aux options Basic et Advanced pour obtenir des compléments d'information sur les valeurs courantes.

3.1.7 Les autres requêtes (Others Menu)

Comme dans les fonctions d'information, l'appel à cette option élargit la zone de contexte pour y ajouter les valeurs courantes. La zone de travail a également un contenu fonction des informations disponibles. Il s'agit des diverses requêtes évoluées.

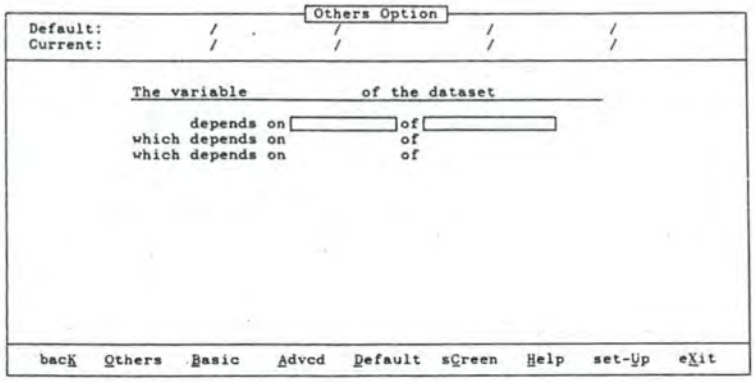


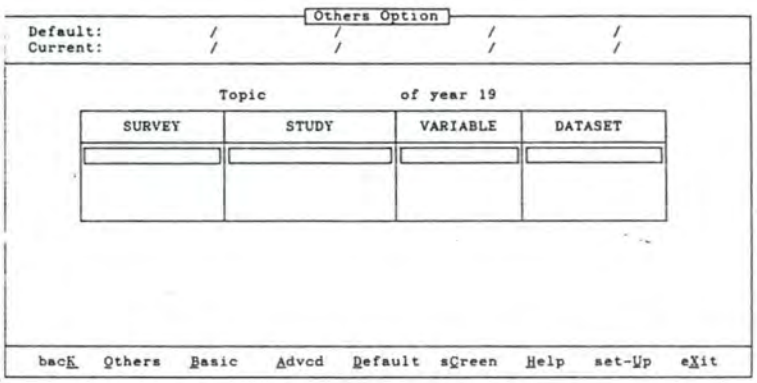
Cette option propose ensuite un écran avec les différents champs résultats demandés. Comme dans l'option de liste, la valeur courante, sur laquelle l'utilisateur peut demander des informations, est en affichage inverse. De même,

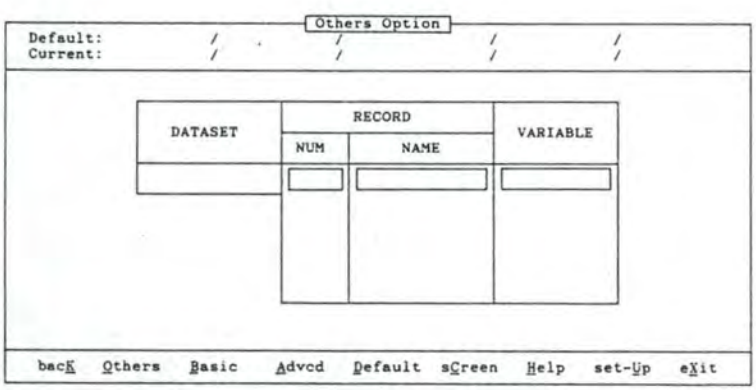
il peut se déplacer dans la liste avec les flèches.

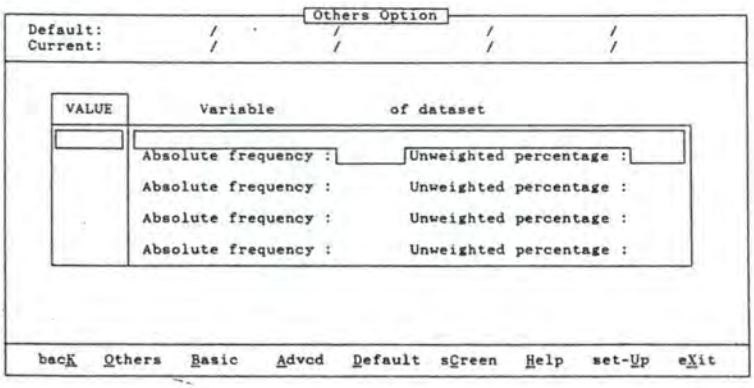
Comme pour les fonctions d'information, il est possible soit de revenir au menu appelant soit de revenir au menu de l'option Others.

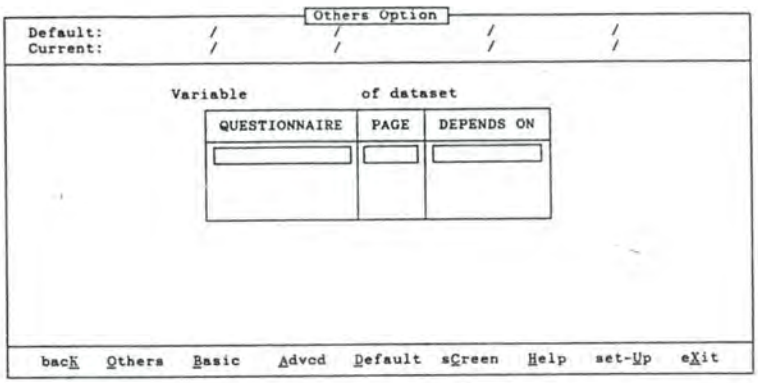
Notons que les fonctions 2, 3 et 5 n'ont finalement pas été implémentées pour des raisons d'incomplétude de la base de données. Voici les écrans de résultat des autres fonctions.


Others Option
Default: / / / /
Current: / / / /
The variable of the dataset
depends on of
which depends on of
which depends on of
bacK Others Basic Advcd Default sCreen Help set-Up eXit


Others Option
Default: / / / /
Current: / / / /
Topic of year 19
SURVEY
STUDY
VARIABLE
DATASET
bacK Others Basic Advcd Default sCreen Help set-Up eXit

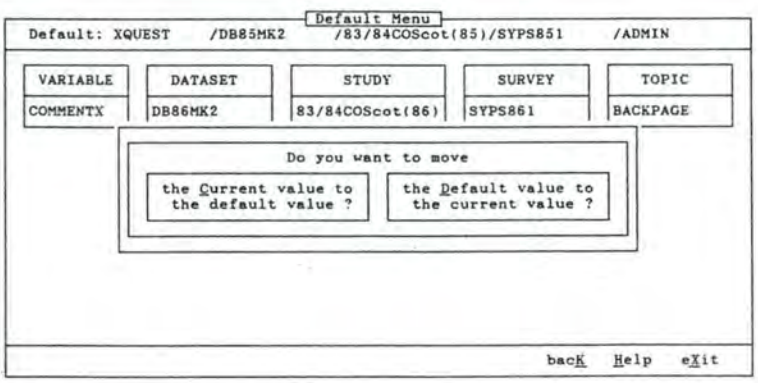

Others Option
Default: / / / /
Current: / / / /
DATASET
RECORD
NUM
NAME
VARIABLE
bacK Others Basic Advcd Default sCreen Help set-Up eXit





3.1.8 Mémorisation des valeurs (Default Option)

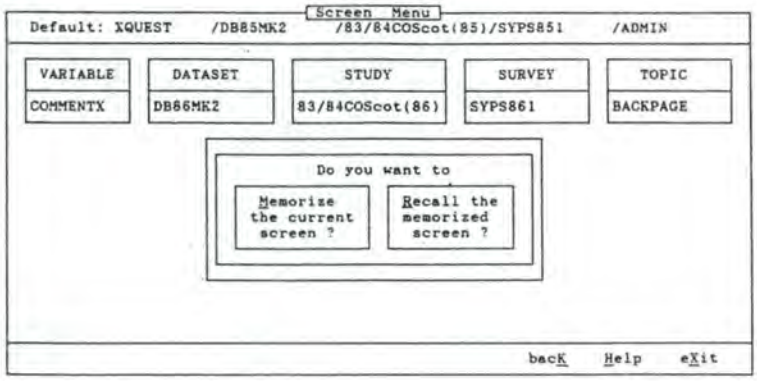
Cette option permet à tout moment d'enregistrer les valeurs courantes contrastées dans l'espace prévu à cet effet dans la zone de contexte.



Cette copie se fait instantanément sauf si l'utilisateur se trouve dans le menu principal. Dans ce dernier cas, une boîte de dialogue apparaît au milieu de la zone de travail. Cette boîte offre deux options : soit mémoriser les valeurs courantes du menu principal, soit restituer les valeurs mémorisées.

3.1.9 Mémorisation d'écran (Screen Option)

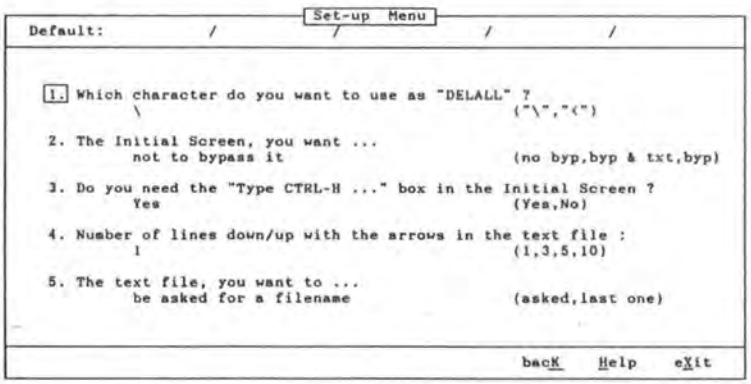
Cette option permet de mémoriser à volonté un écran. Lorsque l'utilisateur y fait appel, une boîte de dialogue apparaît au centre de la zone de travail. Cette boîte propose deux alternatives : soit mémoriser l'écran tel qu'il est sans la boîte de dialogue, soit rappeler un écran précédemment mémorisé et, après l'avoir consulté, reprendre la session là où elle avait été interrompue. Par exemple, le chercheur peut mémoriser une page dans l'option texte et la consulter pendant qu'il effectue des recherches dans DOCDB ou mémoriser une page de fréquences absolues dans les recherches évoluées et la consulter en parcourant son fichier SPSS.



3.1.10 Le menu de paramétrage (Set-up Menu)

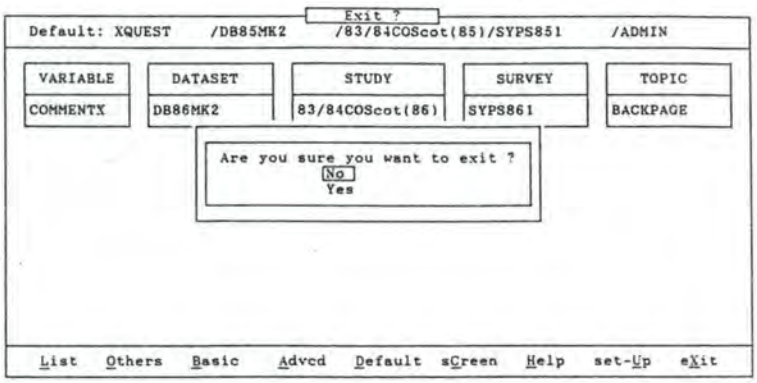
Voici les différents paramètres que l'utilisateur peut déterminer.

Le premier et le quatrième paramètres s'expliquent d'eux-mêmes. Les trois autres déterminent le début de la session. En effet, le deuxième paramètre permet d'éviter le menu initial en se branchant directement soit sur le menu principal soit sur l'option de texte. Le troisième permet de supprimer le message d'aide de l'écran initial. Le dernier paramètre précise si le nom du fichier texte doit être demandé au début de la session ou si le fichier utilisé lors de la précédente session doit être chargé.



3.1.11 La sortie (Exit Menu)

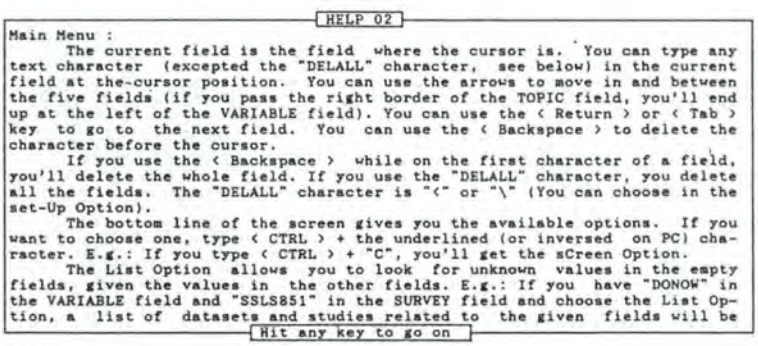
Si l'utilisateur appelle cette option alors qu'il est dans Docdb et qu'il utilise un fichier texte en parallèle, il se retrouve instantanément dans l'option de texte.



Dans les autres cas, une boîte de dialogue apparaît dans la zone de travail. Son but est de confirmer la demande de sortie.

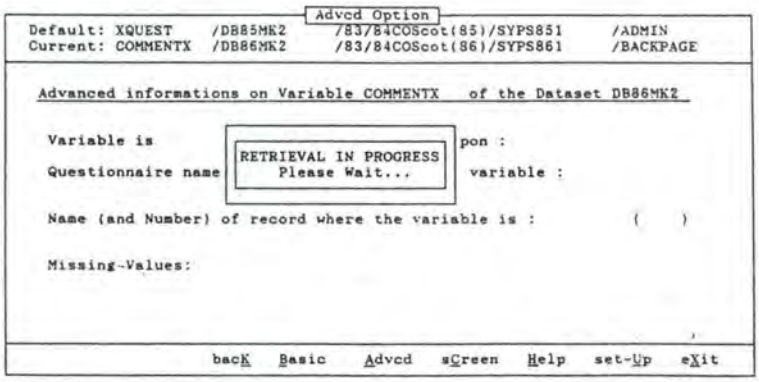
3.1.12 L'aide (Help Screens)

Une ou plusieurs pages d'aide sont disponibles dans tous les écrans. L'utilisateur y trouve expliqués tous les concepts présents à l'écran appelant, principalement les fonctions disponibles dans la barre de menu et leurs effets.

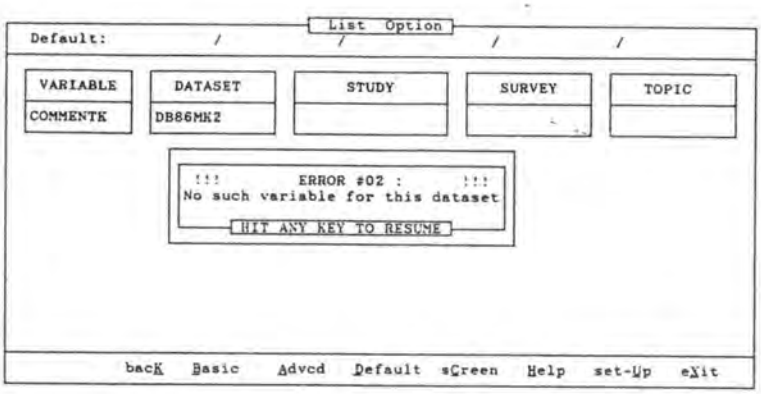


3.1.13 La gestion d'erreurs et les messages d'attente

Pour les requêtes pouvant excéder quelques secondes, l'utilisateur se voit demander de patienter par un message apparaissant au centre de la zone de travail.



De même, lorsque l'utilisateur commet une erreur (ou lorsqu'une erreur de cohérence de la base de données est détectée), un message accompagné d'un signal sonore explique le problème à l'utilisateur.



3.2 Analyse ergonomique

Nous avons essayé au maximum de réaliser un programme qui répondait aux besoins exprimés par les utilisateurs et d'atteindre une utilisation la plus ergonomique possible. Notons cependant que cette interface comporte quelques lacunes en matière d'ergonomie. Celles-ci sont dues au peu de temps qui nous était imparti pour à la fois spécifier et implémenter le prototype.

Nous allons d'abord voir comment nous avons tenté de maximiser le degré d'implication directe de l'utilisateur dans l'application. Nous allons ensuite procéder à

une analyse ergonomique détaillée du prototype sur base des critères ergonomiques empiriques de design vus au cours de Fr. Bodart.[1]

3.2.1 Implication directe de l'utilisateur

Une interface peut être construite soit sur la métaphore de la conversation, soit sur la métaphore du mini-monde. Dans la première, "le véhicule de l'interface est un langage dans lequel l'utilisateur et le système ont une conversation au sujet d'un monde supposé mais non explicitement représenté" [2]. Au contraire, la métaphore du mini-monde suppose que "l'interface devient lui-même un monde dans lequel l'utilisateur peut agir; le monde change d'état en réponse aux actions de l'utilisateur. L'interface en tant que telle disparaît, elle devient invisible"[3].

Nous avons vu au point 2.2.2 que les chercheurs préféraient un dialogue sur base de menus plutôt que sous la forme d'un langage de commande. Nous nous sommes donc orientés vers une construction de l'interface sur base de la métaphore du mini-monde.

Fr. Bodart précise que les conditions requises pour fournir un sentiment d'implication directe sont :

- minimiser les distances sémantique et articulatoire;
- les langages d'input et d'output doivent être inter-référentiels;
- le système doit procurer une réponse immédiate;
- l'interface ne doit pas interférer avec les actions de l'utilisateur.

Détaillons maintenant chacune de ces conditions.

Minimiser les distances sémantique et articulatoire

Précisons que la distance sémantique est la "distance entre l'intention de l'utilisateur et la signification des expressions (dans un langage donné)"[4]. La distance articulatoire est "la distance entre les significations des expressions du langage et leur

---

[1] [BODART89]

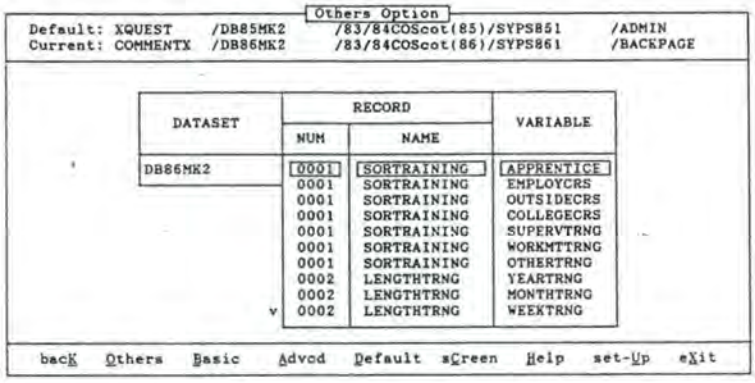
[2] [BODART89],pg18

[3] [BODART89],pg18

[4] [BODART89], pg 14

forme physique"[5]. Pour illustrer ces définitions, prenons l'exemple d'un utilisateur de traitement de texte qui désire mettre un paragraphe en italique. La distance sémantique est l'écart qui existe entre cette intention et la possibilité de la réaliser (par exemple, l'existence de l'option "mettre en italique" dans la barre de menu). D'autre part, la distance articulatoire est fonction de la complexité de la manipulation nécessaire à l'utilisateur pour mettre en italique le paragraphe.

Nous avons essayé autant que possible de réduire la distance sémantique. Prenons l'exemple de l'écran suivant.



Le résultat de cette requête est exprimé dans le langage du chercheur : les noms d'output utilisés font référence aux termes connus par le chercheur.

Dans la réalité, le chercheur, en parcourant la liste de variables, peut se demander quelle est l'étiquette de la variable et si elle est filtrée. De même, l'utilisateur a alors la possibilité pour chaque ligne de l'écran de faire appel instantanément aux fonctions Basic et Advanced.

Le chercheur peut également désirer parcourir sa documentation avec à côté de lui la page sur laquelle il souhaite des renseignements. Pour ce faire, l'utilisateur peut consulter à tout moment l'écran mémorisé à l'aide de la fonction Screen.

Toujours pour minimiser la distance sémantique, une aide contextuelle est disponible en permanence. De plus, la gestion d'erreur est non destructive, c'est-à-dire que l'utilisateur peut toujours reprendre à l'endroit où il a commis une erreur.

Nous avons également essayé de réduire la distance articulatoire aux niveaux syntaxique et lexical. Comme nous l'avons dit au point 3.1, l'utilisateur choisit une option avec la combinaison <CTRL>+<lettre adéquate> au lieu d'une touche de

---

[5] [BODART89], pg 14

basculement vers la barre de menu où il opère alors le choix. Des mots brefs et représentatifs ont été choisis pour la barre de menu et la zone de contexte. Dans une liste, la valeur courante est contrastée, ce qui fournit à l'utilisateur un bon moyen pour se repérer. Pour cette même raison, la lettre de sélection d'une option est en majuscule et soulignée.

Les langages d'input et d'output doivent être inter-référentiels

Cela signifie que la même entité doit pouvoir être utilisée comme input et comme output. Par exemple, dans l'option List, les mêmes champs servent à la fois à l'acquisition des données et à l'affichage des résultats.

Le système doit procurer une réponse immédiate

Notre prototype satisfait à cette condition dans la mesure où toutes les requêtes fournissent soit une réponse immédiate, soit une réponse légèrement différée avec un message invitant l'utilisateur à patienter.

L'interface ne doit pas interférer avec les actions de l'utilisateur

Cette interférence se présente sous la forme de messages d'erreur ou d'aide intempestifs, ou d'interruption pour procéder à des sauvetages de données. Nous respectons cette exigence à une exception près. En effet, la nécessité en PQL d'utiliser, pour le stockage des résultats des requêtes, une base de données intermédiaire de taille limitée, nous contraint à purger celle-ci de temps à autre pour ne pas dépasser sa capacité. Bien sûr, un message en avertit l'utilisateur mais une légère interférence est néanmoins créée dans le déroulement de la tâche.

3.2.2 Etude sur base des critères ergonomiques empiriques de design[6]

"Ces critères parfois qualifiés de règle d'or sont mis en évidence par l'expérimentation"[7].

---

[6] [BODART89], pg 29-37

[7] [BODART89], pg 29

R1 Lutte pour la cohérence

L'idée est de garantir l'unité des éléments de l'interface en éliminant les exceptions et les contradictions.

R1.1 Cohérence et spécification du plan d'action

Si l'application contient des sous-tâches communes, la suite des commandes à utiliser devrait également être commune.

Dans notre prototype, cette cohérence apparaît entre les options List et Others. L'utilisateur peut, dans les deux sous-tâches, parcourir la liste des résultats avec les flèches. La ligne courante est, de part et d'autre, la ligne contrastée et l'utilisateur peut, à tout moment, en demander les informations de base et avancées.

De la même façon, le plan d'action des options Basic et Advanced est tout à fait identique.

Signalons que le choix du nom de fichier dans l'option Getfile aurait pu se faire dans une boîte de dialogue spécifique plutôt que dans la zone de contexte (comme dans l'option Goto).

R1.2 Cohérence d'exécution

Cohérence syntaxique

Tout au long de l'application, l'utilisateur fait appel aux options de la barre de menu ou de la boîte de dialogue à l'aide de la combinaison <CTRL>+<lettre majuscule soulignée dans le mot clé>.

Les options non disponibles ne sont pas affichées. Les autres options sont regroupées pour éviter qu'une commande n'échappe à l'utilisateur si elle est séparée des autres par un grand espace.

Remarquons que dans le menu initial, même si le mode de sélection d'option est cohérent avec les autres, les deux options "Only" et "Text" se trouvent dans la zone de travail mais pas dans la barre de menu, ce qui peut perturber l'utilisateur.

Notons une incohérence syntaxique au niveau de la boîte de dialogue de l'option Exit. En effet, la confirmation se fait à l'aide des flèches et du <RETURN> au

lieu de la séquence <CTRL>+<lettre>. Ceci se justifie par le besoin de faire réfléchir l'utilisateur et par le fait que la combinaison CTRL+Y provoque l'interruption du programme en VMS et que PQL ne permet pas d'inhiber cette séquence.

Cohérence lexicale

Dans notre programme, nous utilisons les termes auxquels les chercheurs sont habitués. En ce qui concerne les mots clés utilisés dans la barre de menu, nous avons choisi des mots brefs et représentatifs. Notons cependant que, bien que le mot other puisse être pris sous la forme substantive, "Other" serait préférable à "Others".

Des termes identiques sont employés dans les différents menus pour désigner les mêmes actions.

Cohérence spatiale

Le balayage de l'écran se faisant en Z, les informations importantes doivent se trouver à partir du coin supérieur gauche ou au centre de l'écran alors qu'il y a lieu de renvoyer les informations moins importantes en bas de l'écran. Nous avons respecté cette cohérence en plaçant le titre et la zone de contexte dans la partie supérieure de l'écran et la zone de travail au centre de l'écran. Notons cependant que la barre de menu aurait dû être placée juste en dessous de la zone de contexte plutôt que dans le bas de l'écran.

R1.3 Cohérence inter-référentielle

Nous l'avons déjà évoquée au point 3.2.1.

R2 Concision

Les dispositifs de communication doivent être brefs et significatifs.

R2.1 Concision et abréviation

Nous avons choisi des termes brefs et expressifs pour faciliter le travail des utilisateurs (réduire la distance sémantique). La seule abréviation que nous ayons utilisée est "Advcd" au lieu de "Advanced" pour l'option d'information avancée.

R2.2 Concision et macro-commandes

Ce critère est sans objet dans notre cas.

R2.3 Concision par défaut

Nous avons utilisé autant que possible les valeurs par défaut. Par exemple, si, après avoir consulté le fichier texte, l'utilisateur revient au menu principal, ce dernier contient par défaut les dernières valeurs utilisées. Rappelons également que l'option Goto du menu texte possède un numéro de ligne par défaut (voir 3.1.3). Le menu Set-up permet également de fixer et de modifier certaines valeurs par défaut.

R3 Structuration des activités

"La structuration des activités concerne l'organisation de l'espace de travail de l'utilisateur"[8].

R3.1 Structuration des commandes par couche de complexité croissante

Il ne nous paraît pas intéressant de réaliser un sous-ensemble utile ou un mode expert (exceptés les raccourcis, voir plus loin) pour notre programme, vu la simplicité des actions.

R3.2 Structuration des commandes par classe de fréquence d'utilisation

Rappelons que les requêtes de fréquences différentes ont été regroupées dans des options différentes (Basic & Advanced, List, Others).

R3.3 Structuration des commandes en tâches fermées

L'utilisation de chacune des options se présente sous la forme d'une sous-tâche fermée, c'est-à-dire que, une fois la consultation terminée, l'utilisateur se retrouve dans la situation précédant l'appel.

R3.4 Granularité des commandes

Il nous semble que la concision choisie pour les commandes ainsi que la hiérarchie des menus correspondent à une granularité optimale.

---

[8] [BODART89],pg 32

R3.5 Valeurs par défaut

Dans notre cas, il parait absurde de parler de valeurs par défaut pour les commandes.

R3.6 Disposition spatiale des commandes

Nous avons déjà évoqué l'utilisation de termes concis pour éviter la surcharge. Rappelons également le défaut concernant le respect de la règle du balayage en Z.

R4 Retour d'information

Il est important d'informer l'utilisateur sur l'état du système.

R4.1 Informer pour rassurer

Dans notre prototype, il s'agit principalement de la gestion du temps de réponse : un message invitant l'utilisateur à patienter apparaît pour les requêtes atteignant et dépassant les quatre secondes. Signalons que nous aurions pu ajouter un indicateur de progression.

R4.2 Informer sur l'espace de travail

Le titre de l'écran permet à l'utilisateur de se situer. Le curseur dans le menu principal ou la ligne contrastée dans les manipulations de listes, le numéro de ligne dans l'option texte sont autant de mécanismes qui permettent à l'utilisateur de se repérer dans l'espace de travail.

R4.3 Informer sur les options

La barre de menu n'est pas figée, son contenu est déterminé par la situation dans laquelle se trouve l'utilisateur. Notons également que les menus des options d'information sont constitués de façon dynamique : seules les informations pertinentes sont disponibles et affichées.

R4.4 Informer sur l'espace du problème

La zone de contexte présente la valeur des entités mémorisées au moyen de l'option Default. Dans certaines options où cela s'avère nécessaire, la zone de contexte contient également la valeur courante des entités du menu principal.

R5 Gestion des erreurs

R5.1 Signalement

Les erreurs sont signalées immédiatement sous une forme standard (numéro et libellé). L'erreur est affichée dans une boîte en surimpression au centre de la zone de travail. Bien que, pour la plupart des erreurs, le libellé soit suffisamment explicite, nous aurions pu rédiger un texte plus complet pour certaines erreurs. Notons que le numéro d'erreur permet à l'utilisateur de se référer à une documentation plus complète dans le manuel.

R5.2 Défaire et refaire

Les actions ne sont pas réversibles mais cela ne porte pas à conséquence puisque, après l'exécution de la tâche, c'est-à-dire la requête, l'utilisateur se retrouve dans la situation dans laquelle il était auparavant.

R6 La flexibilité

La seule flexibilité que nous ayons eu le temps d'aborder est l'adaptation à l'évolution cognitive de l'utilisateur, c'est-à-dire le paramétrage disponible dans l'option Set-up.

Chapitre 4

Feed-back du prototype

Nous ne développerons pas ici une justification de la réalisation d'un feed-back, car elle découle des raisons pour lesquelles nous avons choisi de réaliser un prototype plutôt qu'un programme fini (voir point 2.1).

Nous allons tout d'abord, dans ce chapitre, présenter notre idée de départ. Nous verrons ensuite comment la réalité du CES l'a modifiée pour enfin terminer par la présentation des résultats et une brève conclusion.

4.1 Idée de départ

Nous pensions à l'origine réaliser le feed-back au niveau des besoins et au niveau de l'ergonomie. Par feed-back sur les besoins nous entendons découvrir quels besoins des utilisateurs n'ont pas été rencontrés par le prototype. En effet, les chercheurs, après avoir utilisé celui-ci, peuvent trouver de nouveaux besoins ou affiner ceux qu'ils ont exprimés initialement. Le feed-back au niveau de l'ergonomie permet d'autre part de déterminer les outils d'interaction qui pourraient être ajoutés pour améliorer l'agrément d'utilisation.

Il était prévu qu'une fois le prototype terminé, les chercheurs et les programmeurs l'utilisent pendant plusieurs mois. Le feed-back se serait alors présenté sous la forme d'un questionnaire et d'une discussion informelle. Le questionnaire se serait concentré sur des questions précises concernant le déroulement de la session. Par exemple, les écrans d'aides sont-ils compréhensibles ? Que faudrait-il ajouter dans le Set-up ? Y a-t-il des fonctions de recherches évoluées que vous désireriez voir ajoutées ? La discussion informelle nous aurait permis de connaître les sentiments de l'utilisateur face à l'outil.

4.2 Problèmes rencontrés

Deux problèmes se sont posés : la non-utilisation du prototype et le fait que la base de données n'ait pas été complétée.

La réalité du CES ne nous a pas permis de mener notre feed-back comme nous l'entendions initialement. Un départ de personnel imprévu nous a obligé à réviser notre stratégie. En effet, les deux programmeurs à qui nous avions appris à manipuler le prototype ont quitté le CES, ce qui a empêché la mise sur pied de séances de formation des chercheurs. Le feed-back ne pouvait donc plus se faire sur base d'une utilisation intensive.

De ce fait, il n'était possible de faire le feed-back qu'au niveau des besoins. Nous avons décidé de faire plusieurs séances d'environ une heure avec des groupes de deux ou trois chercheurs, au cours desquelles une première partie était dévolue à la présentation du fonctionnement du prototype et une seconde à une discussion informelle. Une séance a également été réalisée avec l'ensemble des programmeurs.

Le départ des programmeurs affectés au projet Docdb a de même mis un frein au remplissage de la base de données de documentation. De ce fait, l'utilisation du prototype était difficile car beaucoup de requêtes n'aboutissaient pas ou donnaient des résultats incomplets. Cela nous a obligé à chercher un sous-ensemble de la base de données que nous pouvions considérer comme suffisamment complet et cohérent et à n'utiliser que ce sous-ensemble dans les séances avec les chercheurs.

4.3 Résultats du feed-back

Nous allons décomposer ces résultats en deux groupes : d'une part les résultats touchant aux besoins de l'utilisateur et d'autre part les résultats concernant l'ergonomie.

4.3.1 Nouveaux besoins

Quatre nouveaux besoins se sont dégagés de nos discussions avec les chercheurs et les programmeurs.

La mise à disposition d'un véritable éditeur de texte plutôt qu'une simple visualisation de texte répondrait à un besoin exprimé par plusieurs chercheurs. En effet, ceux-ci souhaiteraient pouvoir rédiger un texte, annoter les listings sur lesquels

ils travaillent, etc ... Ce nouveau besoin pourrait nous conduire à repenser notre option texte suivant deux pistes alternatives : utiliser un éditeur déjà existant ou en écrire un. Nous ne savons pas si PQL permet la commutation entre deux applications indépendantes (l'accès à Docdb et l'éditeur EMACS par exemple) mais, si cela était réalisable, cela permettrait aux chercheurs d'utiliser le traitement de texte auquel ils sont habitués. La réalisation d'un nouvel éditeur au sein de l'application serait évidemment plus coûteuse, mais permettrait de conserver une cohérence entre les deux sous-tâches de l'application.

Le nouveau besoin le plus important que nous ayons relevé est la possibilité de déterminer, dans les recherches de base (l'option List), quels seraient les champs de résultat souhaités à partir des champs donnés. Rappelons que pour l'instant ces champs ont été fixés a priori à partir des besoins exprimés au départ par les chercheurs et ne sont pas adaptables. Nous pourrions imaginer une solution où les utilisateurs une fois leurs champs en entrée définis, désigneraient par un moyen quelconque les champs qu'ils désirent voir se dérouler au moyen de l'option List. Notons que les combinaisons par défaut existantes seraient toujours disponibles si rien n'est spécifié. Certaines requêtes pourraient prendre un temps non négligeable, notamment celle qui consiste à lister les sujets si rien n'est connu (ceci est un besoin énoncé par les chercheurs).

Une demande qui nous a été réitérée est celle de pouvoir imprimer les renseignements obtenus via l'application. Rappelons que nous n'avons pu réaliser cette fonctionnalité par manque de temps.

Un besoin d'importance secondaire est la possibilité de trouver une chaîne de caractères dans le fichier texte. Ce besoin sera rencontré lorsque l'on s'attellera à résoudre le problème de l'éditeur. En effet, si un traitement de texte existant est utilisé, il comporte habituellement cette fonction, sinon elle sera reprise dans la solution implémentée.

4.3.2 Nouveaux outils

Nous avons également pu relever quatre outils qui amélioreraient l'ergonomie du logiciel.

Signalons que certains datasets ont leurs variables préfixées ou postfixées d'une lettre. Par exemple, les variables COMMENT et COMMENTX ou les variables JOBFATH ou JBFATHER recouvrent le même concept dans deux datasets différents. Pour y accéder, l'utilisateur désirerait disposer du concept de joker, par exemple l'étoile

et le point d'interrogation comme dans la plupart des systèmes d'exploitation. Ce mécanisme ne serait disponible que pour l'option List.

L'option Getfile du menu Text (voir point 3.1.3) peut être améliorée : une liste des fichiers du répertoire courant, ainsi qu'un mécanisme de navigation à travers les différents répertoires, pourraient être ajoutés. De même, une liste des derniers fichiers consultés pourrait y être jointe.

Il conviendrait également de fournir des raccourcis pour accéder par exemple directement aux informations de base sur la variable, sans passer par le menu de l'option Basic, ou aux fréquences d'une variable sans passer par le menu de l'option Others. Un autre type de raccourcis pourrait être utilisé pour accéder, via leur première lettre, aux éléments d'une longue liste dans les options List et Others.

Remarquons pour terminer que l'ordre des entités de base dans le menu principal pourrait être amélioré aux yeux des utilisateurs en permutant le dataset et la variable.

4.4 Conclusions

Dans l'ensemble, les chercheurs sont enthousiastes et, malgré quelques remarques constructives, ils sont déjà prêts à utiliser le prototype dans son état actuel. Une recherche n'est en aucun cas plus longue avec le logiciel qu'à l'aide des dictionnaires papiers. Certains chercheurs estimaient même que les fréquences de l'option Others leur permettraient de réaliser le travail d'une demi-journée en quelques minutes.

Notons cependant que si le logiciel est utilisable dans l'état actuel, ce n'est pas le cas de la base de données. Une fois celle-ci remplie, il sera possible et utile de réaliser un nouveau feed-back qui comportera de nouveaux éléments suite à une utilisation intensive de l'application.

Deuxième Partie

Bases théoriques : Alternatives et choix

La deuxième partie de ce travail fournit un cadre théorique qui nous permettra de spécifier l'application d'interface.

L'ingéniérie logicielle nous apprend l'existence de nombreuses approches de conception. Un choix doit être fait, privilégiant ainsi une approche plutôt qu'une autre. Il doit être mûrement réfléchi et adapté au problème abordé.

C'est pourquoi nous allons, dans un premier chapitre, nous attarder à préciser les qualités déterminantes pour qu'un logiciel soit bon.

Nous présentons ensuite les deux approches qui peuvent être aujourd'hui considérées comme classiques : la conception fonctionnelle et la conception orientée objets.

Pour parfaire et surtout garantir l'adéquation de notre choix, nous étudions dans un troisième chapitre les facteurs qui influencent notre décision et procédons au choix.

Nous terminons en présentant le langage de conception que nous avons retenu.

Chapitre 5

Les qualités d'un programme

5.1 Les qualités de base [1]

Avant toutes choses, il convient de nous interroger sur les facteurs qui font qu'un programme est de bonne qualité. En effet, l'objectif de l'ingénierie logicielle est d'aider à produire un logiciel de qualité tout en garantissant celle-ci.

La qualité la plus importante est évidemment l'adéquation. Nous entendons par cela que le programme accomplisse une tâche qui rencontre tous les besoins et spécifications. Si un problème ne respecte pas cet impératif d'adéquation, il est sans objet. Bien que cette qualité soit essentielle, il est à remarquer que cet objectif est diversement atteint. La meilleure manière de s'assurer de l'adéquation est d'exprimer les besoins de façon formelle et ainsi de disposer d'un moyen de contrôle de la concordance entre besoin et résultat.

Une deuxième qualité est la fiabilité (ou robustesse) c'est-à-dire la capacité qu'a le programme de réagir à des événements anormaux. Il convient dans ce cas de s'assurer que le programme garantit que la cohérence du système (les données principalement) est conservée.

L'extensibilité est la troisième qualité qu'un programme devrait posséder. Il s'agit de construire celui-ci de manière telle qu'il puisse s'adapter aisément aux changements de spécifications. Par ceux-ci, nous entendons autant les changements dus aux corrections des spécifications existantes qu'à l'adjonction de nouvelles spécifications résultant de nouveaux besoins.

La quatrième qualité, la réutilisabilité, est liée à l'extensibilité. Par réutilisabilité, nous entendons l'aptitude d'un programme à servir, en tout ou en partie, pour le développement de

---

[1] Ce chapitre est basé sur [Meyer88] et [Sommerville89].

nouvelles applications. Notons que ces qualités d'extensibilité et de réutilisabilité appellent une architecture flexible et modulaire. Nous reviendrons plus loin sur la notion de modularité.

Une autre qualité d'un programme est l'efficacité. Nous n'entendons pas par là le fait d'utiliser la moindre des fonctionnalités proposées par le hardware pour obtenir un gain de performance mais plutôt de garantir une utilisation optimale des ressources disponibles en ne sacrifiant pas la portabilité au profit de la vitesse d'exécution. Il est d'ailleurs à remarquer qu'utiliser des fonctionnalités trop spécifiques du hardware entrerait en contradiction avec les notions d'extensibilité et de réutilisabilité.

Une dernière qualité importante est d'avoir une interface-utilisateur appropriée. Un grand nombre de software ne sont en effet pas utilisés de façon optimale car leur interface est difficile.

Ces six qualités fondamentales d'un programme ne constituent pas une liste exhaustive. Nous pouvons aussi citer la compatibilité (facilité avec laquelle un programme peut être associé avec d'autres), la portabilité (capacité de s'exécuter sur d'autres hardware avec un minimum de modifications), la vérifiabilité (facilité d'intégrer des tests pour découvrir des erreurs) et l'intégrité (capacité de se protéger contre des accès non autorisés).

Remarquons qu'il n'est pas possible de satisfaire à toutes ces qualités en même temps. Il convient donc de faire des choix comme, par exemple, de favoriser la réutilisabilité, qui appelle une réflexion plus large que le problème abordé, aux dépens de l'efficacité, qui met l'accent sur une résolution la plus efficace possible du problème.

5.2 La modularité

Revenons maintenant à la modularité, à l'origine des deux qualités d'un programme que sont l'extensibilité et la réutilisabilité. La modularité permet le développement de programme constitué d'éléments autonomes reliés par une structure simple et cohérente.

B.Meyer[2] propose cinq principes que doit respecter un programme pour avoir une bonne modularité :

- Les modules doivent correspondre à des unités syntaxiques du langage utilisé (principe de l'unité linguistique);

---

[2] [Meyer88]

- Chaque module doit communiquer avec le moins de modules possible (principe du peu d'interfaces);
- Si deux modules doivent communiquer, il faut qu'ils échangent un minimum d'information (principe du couplage faible);
- Si deux modules communiquent, cela doit se voir en regardant le texte des deux ou de l'un des deux (principe de l'interface apparent);
- Toute information sur un module doit être limitée à ce module si elle n'est pas spécifiquement déclarée publique (principe de la dissimulation d'information (information hiding) de Parnas[3])

Selon Parnas[4], les bénéfices d'une approche modulaire sont :

- un raccourcissement du temps de développement;
- une meilleure flexibilité du résultat;
- une amélioration de la compréhension.

Le système dans son ensemble peut dès lors être mieux conçu car il est mieux compris.

---

[3] [Parnas72]

[4] [Parnas72], pg 1054

Chapitre 6

Les différentes approches

Attardons nous maintenant sur la présentation de deux approches possibles pour satisfaire aux qualités dont nous avons parlé au chapitre précédent : l'approche fonctionnelle et l'approche orientée objets. Nous présenterons chacune des approches avant d'envisager une éventuelle complémentarité entre elles.

6.1 L'approche fonctionnelle

Dans cette approche, la cohésion de la conception se fait autour de la notion de fonction.

Bien que l'approche fonctionnelle n'ait été théorisée qu'au début des années septante, elle est en fait utilisée depuis les débuts de la programmation.

Comme Wirth[1] l'a souligné, " au début, l'attention est dirigée principalement vers les problèmes globaux (...). Comme le processus de conception progresse, le problème est divisé en sous-problèmes et, graduellement, plus de considération est apportée aux détails de la spécification du problème (...)".

Baudouin et Meyer encouragent une telle approche car elle permet de " séparer les difficultés pour diminuer le niveau de complexité des problèmes à résoudre "[2].

Le fait de considérer d'abord le problème global puis de le subdiviser est appelé approche descendante (top-down).

Donc, l'approche fonctionnelle consiste à considérer un système comme un ensemble d'unités fonctionnelles interactives. Celles-ci peuvent être vues comme des traitements dont on connaît les flux en entrées et en sorties.

---

[1] [Wirth73], pg 125

[2] [Baudouin78], pg 510

De cette manière, la conception d'un système peut être décrite par les étapes suivantes :

- examiner les flux en entrée et en sortie;
- déterminer les transformations nécessaires pour définir ces flux;
- soit la transformation observée est une fonction élémentaire, soit elle est décomposable en sous-fonctions pour lesquelles on réapplique les différentes étapes.

Nous retiendrons que la difficulté ne réside pas tant dans la description des unités fonctionnelles que dans la définition de leur interaction et l'élaboration d'une stratégie de développement.

DeMarco[3] propose d'adopter la démarche qui a recours aux outils suivants :

- Un diagramme des flux de données (Data flow diagram) pour nous aider à subdiviser le problème et pour documenter cette subdivision. Ce diagramme est un réseau de transformations non nécessairement élémentaires qui décrit les flux de données et les transformations qui y sont appliquées.
- Un dictionnaire de données (Data dictionnary) pour répertorier les descriptions des fonctions ainsi que leur interface.
- De nouveaux outils pour décrire la logique et les critères de décision (par exemple, des tables de décisions).

A coté des diagrammes de flux, il existe également une représentation hiérarchique du système appelée plan de la structure (Structure chart). Celui-ci décrit l'organisation du programme sous forme d'une arborescence de traitements : "Le niveau supérieur montre la division la plus importante de l'application; les niveaux inférieurs subdivise le travail de chacune de ces grandes divisions. La philosophie sous-jacente du système apparaît au sommet et les détails à la base"[4].

Nous pouvons pour terminer relever, à l'instar de Booch[5], que les méthodes fonctionnelles souffrent des limitations suivantes :

- elles ne permettent pas la dissimulation d'information ni l'abstraction, ce qui disperse les modifications dans les divers modules (et augmente leur nombre);

---

[3] [DeMarco79]

[4] [DeMarco79], pg 305

[5] [Booch86], pg 211

- à la différence de l'orientée objets, l'approche fonctionnelle est peu appropriée pour les problèmes concurrents.

6.2 L'approche orientée objets

Dans cette approche, la cohésion de la conception se fait autour de la notion d'objet. Nous donnerons une définition plus précise de cette conception plus loin.

Le principe de la dissimulation d'information (information hiding)[6], que nous avons déjà mentionné au point 5.2, est à la base de la conception orientée objets. Rappelons que la dissimulation d'information consiste à cacher autant d'informations que possible dans les modules issus de la conception.

Dans l'approche orientée objets, le système est vu non plus comme un ensemble de fonctions mais comme un ensemble d'objets interagissants.

Un objet peut être défini comme "un ensemble autonome (self-contained) de variables qui ne peuvent être manipulées que par un ensemble de méthodes définies exclusivement à cet usage"[7]. Notons que certains auteurs donnent une définition similaire en utilisant les mots "services" et "attributs" en lieu et place, respectivement, de "méthodes" et "variables"[8].

Les objets communiquent entre eux à l'aide de messages. Ceux-ci servent à déclencher l'exécution, d'une méthode dans l'objet concerné.

Lors de la phase de conception, pour trouver les objets, il convient de regarder sur quels aspects du réel l'application se base. La difficulté en orienté objets n'est pas de chercher une méthode de résolution du problème mais de choisir judicieusement les objets qui composent cette réalité.

Dans un souci de modularité, il serait souhaitable que la description des données soit assez abstraite. La théorie des types de données abstraits[9] permet cette abstraction en fournissant des spécifications complètes tout en évitant la surspécification.

---

[6] [Parnas72]

[7] [Nelson91], pg 4

[8] Par exemple, [Coad92], pg 44

[9] [Liskov74]

"Un type de données abstrait détermine une classe d'objets[10] abstraits qui est complètement définie par les opérations disponibles sur les objets. Cela signifie qu'un type de données abstrait peut être défini en décrivant les opérations sur ce type"[11].

Les types de données abstraits sont définis de façon mathématique et formelle par des fonctions, préconditions et axiomes. Tous ces concepts constituent la sémantique et sont indispensables pour que la description soit complète et sans ambiguïté.

De ces types de données abstraits, Meyer[12] tire une définition de l'approche orientée objets : "la conception orientées objets est la réalisation d'une application comme une collection structurée d'implémentation de types de données abstraits". En définitive, nous pouvons dire qu'un objet est formé des composantes suivantes : type abstrait de données, identité et méthodes.

Pour être complets, nous devons maintenant définir les divers concepts utilisés dans l'approche orientée objets, à savoir les notions de classes et d'héritage.

Une classe d'objet est un concept statique : si une classe est un type, on peut dire que l'objet est une instance de ce type.

L'héritage "nous permet de définir une nouvelle classe basée sur la définition d'une classe existante (...). Une sous-classe hérite de toutes les variables et méthodes définies dans sa super-classe, peu importe si les méthodes et variables de la super-classe sont elles-mêmes héritées"[13].

Un autre mécanisme pour améliorer la flexibilité d'une application est la généricité. C'est une technique, plus statique que l'héritage, qui consiste à définir les classes d'objet avec des paramètres de types génériques pour éviter de devoir écrire beaucoup de classes quasi identiques. Ces types génériques sont instanciés sous forme de types statiques, ce qui permet de conserver la sûreté garantie par ceux-ci.

Pour terminer, remarquons qu'il ne faut pas confondre la conception orientée objets avec la programmation orientée objets. En effet, il s'agit d'une manière de concevoir une

---

[10] Notons que Liskov emploie ici ce terme au sens courant et non informatique.

[11] [Liskov74], pg 51

[12] [Meyer88], pg 59

[13] [Nelson91], pg 5

application et elle ne présage en rien du langage d'implémentation, celui-ci pouvant être aussi bien fonctionnel qu'orienté objets.

6.3 La complémentarité

Avant d'analyser nos besoins et de choisir l'une ou l'autre approche de conception, il convient, à l'instar de Sommerville, de remarquer qu'il serait absurde d'être dogmatique et de vouloir utiliser l'orienté objets en toutes circonstances quel que soit le système à développer. "A certains niveaux d'abstraction, une vue fonctionnelle est plus facile à dériver des besoins du système qu'une vue orientée objets"[14].

Toujours selon Sommerville, il est possible lors de la conception d'une même application d'utiliser l'une ou l'autre approche à des niveaux différents. Elles sont donc complémentaires.

Cette complémentarité au niveau de la conception apparaît aussi entre la phase de conception et la phase d'implémentation (nous l'avons déjà évoquée au point précédent).

---

[14] [Sommerville89], pg 226

Chapitre 7

Choix d'une approche

Nous allons commencer par nous poser la question de savoir quels sont les facteurs propres au CES qui influencent le choix d'une approche. Nous soulignons ensuite la nécessité de modulariser et discutons des différentes formes de réutilisabilité. Nous terminons en choisissant une approche.

7.1 Les facteurs de décisions

Comme nous avons pu l'entrevoir dans la première partie, une caractéristique remarquable du CES est son contexte mouvant.

Cette instabilité va transparaître à trois niveaux : au niveau des données, au niveau des besoins de l'utilisateur et au niveau du support, tant logique que physique.

D'une enquête à l'autre, le format des données évolue. En effet, puisque la base même des enquêtes du CES est une réalité en changement perpétuel, le format des données est lui-même modifié en conséquence.

De même, la sémantique attachée aux données change. Par exemple, une question à choix multiples comporte une année le choix "ne sait pas", option absente des réponses possibles à une question identique les années précédentes; le champs des réponses possibles est donc altéré et le sens de l'information attachée à cette question également.

Les chercheurs en sociologie, c'est-à-dire les utilisateurs finaux, dont l'environnement de travail est aussi la réalité, déterminent les questions à poser, c'est-à-dire leurs besoins, en fonction de ce que les commanditaires des enquêtes demandent. Leurs besoins évoluent donc dans le temps.

Un facteur très décisif est une possible migration vers un autre support physique. Par exemple, le CES pourrait abandonner les terminaux en mode texte pour des stations de travail graphiques.

Notons également la possibilité de changement de support logique. Par exemple, le CES envisage l'éventualité d'utiliser un nouveau système de gestion de bases de données en lieu et place de SIR.

A coté de ce contexte changeant, nous pouvons relever deux autres facteurs qui peuvent influencer notre choix.

Comme la plupart des centres de recherche universitaires, le CES ne dispose de moyens financiers inépuisables. Il s'agit donc d'adopter des choix économiquement raisonnables.

Pour terminer, nous pouvons constater qu'en quatre années, il y a eu quatre arrivées et autant de départs dans le cadre du projet Docdb. Cette rotation de personnel n'est donc pas négligeable.

7.2 La nécessité de modulariser

Remarquons tout d'abord que le coût des softwares comporte une moyenne de 80% de maintenance et que ce pourcentage pourrait être de 95% en 1995[1].

Observons ensuite le graphique présenté à la figure 7.1.

Ce diagramme montre ce que les coûts de maintenance couvrent réellement. Nous voyons que plus des deux cinquièmes des activités de maintenance sont constitués par des changements des besoins des utilisateurs. 17% sont dus aux modifications de format des données. Notons que ceux-ci sont inévitables mais que le problème réside dans la dispersion à travers tout le programme de ces changements.

Meyer constate que "l'importance de cette proportion [59,2%] semble refléter le manque d'extensibilité de la plupart des applications : les systèmes sont beaucoup plus difficiles à changer qu'ils ne devraient"[2].

La réponse majeure à ces problèmes serait donc une meilleure modularité (que nous avons évoquée au chapitre 5).

---

[1] [Ovum90]

[2] [Meyer88], pg 8

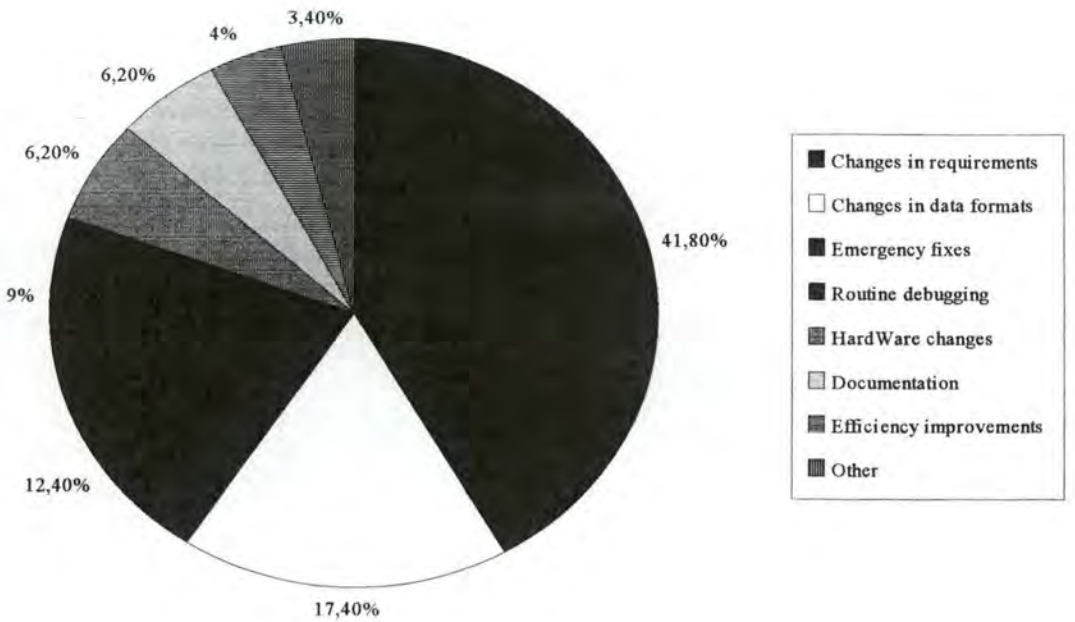


Figure 7.1 - Répartition des coûts de maintenance[3]

En ce qui concerne le CES, aucuns chiffres concernant les coûts de maintenance ne sont disponibles. Mme Lamb pense que ces chiffres sont applicables au CES et qu'il est cependant probable que les proportions de changements au niveau des besoins des utilisateurs et des formats de données soient plus importantes. Nous pensons donc que, dans le cas du CES, la modularisation diminuerait de façon appréciable le temps consacré à la maintenance.

7.3 Réutilisation du code et/ou réutilisation de la conception

La modularité peut être atteinte via deux approches différentes : la réutilisation du code ou la réutilisation de la conception.

L'idée derrière la réutilisation du code est de constituer des librairies de modules précompilés dans un langage de programmation donné et de les utiliser lors de la maintenance ou du développement de nouvelles applications.

---

[3] Source : [Lientz79] corroboré par [Grendle86] et [Casexpo89]

Pour Biggerstaff et Richter[4], cette stratégie pose plusieurs problèmes :

- le gain atteint vite un plafond difficile à franchir;
- la partie du système utilisant les composants est petite (bien moins que la moitié);
- les programmes construits de cette manière ont des problèmes de performance et de fonctionnalité inappropriée.

L'idée de la réutilisation de la conception consiste à créer une réserve de spécifications sur papier décrivant les structures détaillées des applications.

Biggerstaff et Richter considèrent que cette approche pose un problème pratique : comment représenter l'information de conception ? Soit le langage de conception est trop près du langage de programmation et est donc trop spécifique, soit il en est trop éloigné et risque de ne pas évoluer en même temps que l'implémentation.

Meyer[5] s'accommode de cette objection en disant que "la situation est différente pour les approches qui éliminent la frontière entre conception et implémentation", c'est-à-dire l'orienté objets.

Dans le cas du CES, nous sommes d'autant plus enclins à prôner la réutilisation de conception que, à ces arguments, s'ajoute le risque que le support physique et/ou logique ne soit modifié.

Notons pour terminer que les deux approches de réutilisation ne sont pas mutuellement exclusives. Par exemple, si les supports ne changent pas, une réutilisation du code en parallèle à celle de la conception sera possible au CES.

7.4 Orienté objets ou fonctionnel

Nous devons maintenant choisir entre les deux grandes approches de conception présentée au chapitre 6.

Commençons par rappeler qu'une qualité importante d'une application est sa modularité. Nous avons vu que celle-ci est une solution possible pour réduire les coûts de maintenance dus aux changements de besoin de l'utilisateur et aux changements de format des données. Nous savons également que ces deux problèmes sont non négligeables au CES.

---

[4] [Biggerstaff89], pg 8

[5] [Meyer88], pg 31

Meyer et Sommerville[6] sont d'accord pour dire que la mise en pratique des types de données abstraits, les classes d'objets, peuvent être considérés comme des composants modulaires réutilisables. Du point de vue de la modularité, l'approche orienté objets nous semble donc être celle qui convient le mieux.

Un dernier facteur qui peut influencer notre choix est la rotation du personnel au CES. Celle-ci étant relativement importante, il est nécessaire que les nouveaux venus puissent comprendre le plus aisément possible les applications dont ils doivent assurer la maintenance. Cela nous incite également à choisir l'approche orientée objets car "en général, la compréhension et la maintenance sont améliorées par le fait que les objets et leurs opérations associées sont localisées [à ces objets]"[7].

Dans le cas particulier du CES, il nous apparaît donc souhaitable d'adopter une stratégie de réutilisation de conception basée sur une approche orientée objets. Rappelons que cela ne présage en rien du langage ou du type de langage qui sera utilisé à l'implémentation.

---

[6] [Meyer88], page 59 et [Sommerville90] page 358

[7] [Booch86], pg 215

Chapitre 8

OBLOG

Après avoir opté pour une approche orientée objets de la conception, il nous reste à présenter le langage que nous avons choisi pour supporter cette approche. Ce langage s'appelle OBLOG.

8.1 Qu'est-ce qu'OBLOG ?

OBLOG signifie OBject oriented LOGic. C'est à la fois une méthode et un langage de conception. OBLOG est basé sur les théories du professeur A.Sernadas et est développé par la firme portugaise ESDI. Son intention est de concevoir une nouvelle génération de système d'information financier. Elle compte atteindre cet objectif en développant un outil CASE supportant le langage et la méthodologie OBLOG.

Le langage est entièrement graphique (même s'il existe un équivalent textuel), ce qui facilite grandement l'utilisation de l'outil. Notons également qu'OBLOG est un langage défini de manière formelle ce qui garantit sa consistance et sa cohérence. Remarquons aussi qu'OBLOG autorise la concurrence entre objets.

OBLOG permettra de construire une application de la conception globale jusqu'à la génération du code inclue. L'utilisation d'un même langage tout au long du cycle de vie de l'application a les avantages suivants :

- "absence de distance sémantique entre les étapes du développement;
- tout le monde (y compris les analystes) participe au produit final;
- les coûts et les temps de développement sont plus aisés à contrôler;
- s'assurer de la qualité est plus facile;
- la maintenance est faite au bon niveau;

- aucune distance croissante n'apparaît entre les spécifications et le programme;
- les coûts de maintenance sont gardés linéaires"[1].

Nous ne considérerons ici OBLOG que dans son aspect langage de conception et non en tant qu'outil CASE.

8.2 Les concepts de base de l'orienté objets en OBLOG[2]

Comme dans l'approche orientée objets théorique, le concept fondamental d'OBLOG est évidemment celui de l'objet. Un objet OBLOG comporte des événements et des attributs correspondant respectivement aux variables et méthodes de la théorie. De plus, un objet OBLOG est caractérisé par un comportement qui est un enchaînement de ses événements possibles.

Comme nous venons de le dire, l'attribut OBLOG correspond à la notion de variable de la théorie. Il peut être soit constant, c'est-à-dire que sa valeur est fixée à la naissance de l'objet, soit variable, c'est-à-dire que sa valeur change pendant la vie de l'objet. L'attribut variable est également appelé slot. Notons que l'attribut variable peut être soit total soit partiel. Il est partiel lorsqu'il n'est pas défini à tout moment de l'existence de l'objet. Un attribut prend sa valeur dans un codomaine particulier qui est défini soit par un type de donnée, soit par une classe d'objet.

Comme dans la théorie orientée objets, un objet OBLOG est une instance d'une classe, d'un ensemble d'objets semblables. Une classe OBLOG est décrite par un ensemble de diagrammes (dont nous reparlerons au point 8.3). Une classe d'objet peut être soit unique (single), si la classe ne contient qu'un objet, soit ouverte (loose), sinon.

Le concept de méthode dans la théorie orientée objets se nomme événement en OBLOG. Il y a trois types possibles d'événements : l'événement de naissance (birth), de mise à jour (update) et de mort (death). Un événement peut être initié par l'objet auquel il appartient ou par une requête extérieure. Ils sont alors appelés respectivement endogènes (self-initiative) ou exogène (external initiative).

---

[1] [ESDI92b], pg 5

[2] ce point est basé sur [ESDI92c] & [ESDI92d]

L'appel d'événement (event calling) est le concept d'OBLOG correspondant au concept théorique de message. C'est un mécanisme orienté : il y a un événement appelant et un événement appelé.

OBLOG permet l'incorporation d'objets dans d'autres objets. Cette incorporation est une amélioration de la notion d'héritage de l'orienté objets théorique. Contrairement à ce qui ce fait dans celui-ci où les mécanismes d'héritage sont des mécanismes d'héritage de schéma et où aucune information n'est ajoutée, le mécanisme d'incorporation d'OBLOG produit une nouvelle information : si quelque chose se produit dans l'objet composant, il se propage dans l'objet incorporateur en changeant également son état. L'objet incorporateur hérite de tous les attributs et événements de l'objet composant.

8.3 Les concepts complémentaires

En plus des concepts de l'orienté objets traditionnel décrits au point 6.2, OBLOG supporte des concepts qui lui sont propres.

Chaque objet est défini par quatre diagrammes : le diagramme matriciel (matrix diagram), le diagramme de comportement (behaviour diagram), le diagramme d'initialisation (attribute initialization diagram) et le diagramme de mise à jour (attribute updating diagram).

Le diagramme matriciel représente l'objet et l'ensemble des attributs et événements qui lui sont propres. Notons que les attributs de ce diagramme peuvent être soit contraints soit dérivés, c'est-à-dire qu'ils sont respectivement restreints par une condition ou bien dérivés d'autres attributs.

Le diagramme de comportement permet d'exprimer la manière dont s'enchaînent les événements d'un objet ou d'une classe d'objet. L'ensemble se présente sous la forme d'une suite de situation, et le passage d'une situation à une autre se fait via l'occurrence d'un événement. Ce passage peut être conditionnel. Un diagramme de comportement commence par un événement de naissance. Viennent ensuite zéro, un ou plusieurs événements de mise à jour, et finalement, un événement de mort (facultatif).

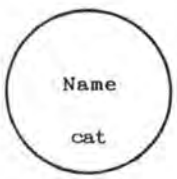
Le diagramme d'initialisation spécifie les valeurs que doivent prendre les attributs à la naissance. L'initialisation des attributs est obligatoire s'ils sont constants et facultatives sinon.

Le diagramme de mise à jour décrit les modifications des attributs pour chaque événement de mise à jour. Les attributs dérivés ne doivent pas être mentionnés dans ce diagramme parce que leur valeur dépend de celle d'autres attributs.

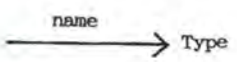
Il existe des diagrammes pour décrire l'appel d'événement entre deux objets. Ce sont des diagrammes d'interaction et de liaison de valeurs (value binding). Le premier détermine quel est l'événement appelant et l'événement appelé et éventuellement à quelles conditions l'appel a lieu. Le second décrit quelles sont les valeurs échangées.

8.4 La notation graphique

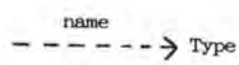
Nous exposons maintenant la représentation graphique des concepts d'OBLOG que nous utiliserons dans la troisième partie.



représente un objet ou une classe d'objet de nom Name. Dans le cas d'une classe d'objet, le paramètre Cat est "1" si la classe est unique, "?" si elle est ouverte.



représente un attribut variable (slot). Name est le nom de l'attribut et Type représente son type.

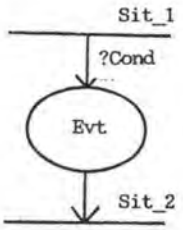


représente un attribut partiel. Name est le nom de l'attribut et Type représente son type.

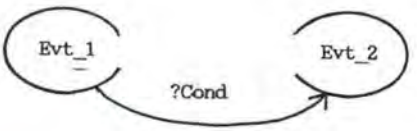
représente un événement. Name est le nom de cet événement. X représente zéro, un ou deux symbole(s) suivant(s) :

- ! si l'événement est endogène;
- * si l'événement est une naissance;
- + si l'événement est une mort.

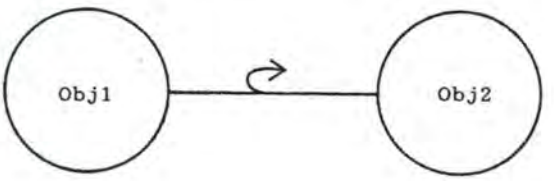
représente une situation dans un diagramme de comportement. Sit_Name, le nom de cette situation, est facultatif.



représente le passage d'une situation Sit_1 à une situation Sit_2 via un événement Evt. Le passage se fait à la condition Cond. Une condition est une formule de premier ordre.



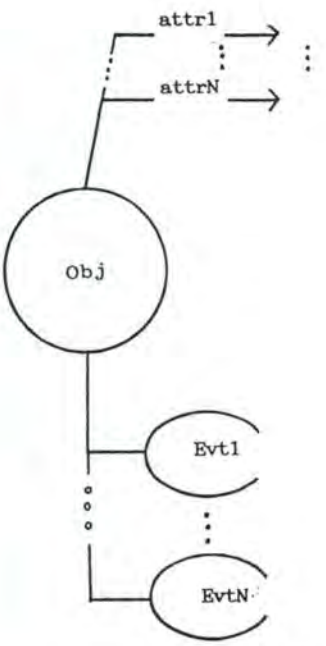
représente l'appel événement. Evt1 est événement appelant et Evt2 l'événement appelé. Cet appel n'a lieu que si la condition Cond est vérifiée.



représente le mécanisme d'incorporation. Obj1 est l'objet composant et Obj2 l'objet incorporateur.

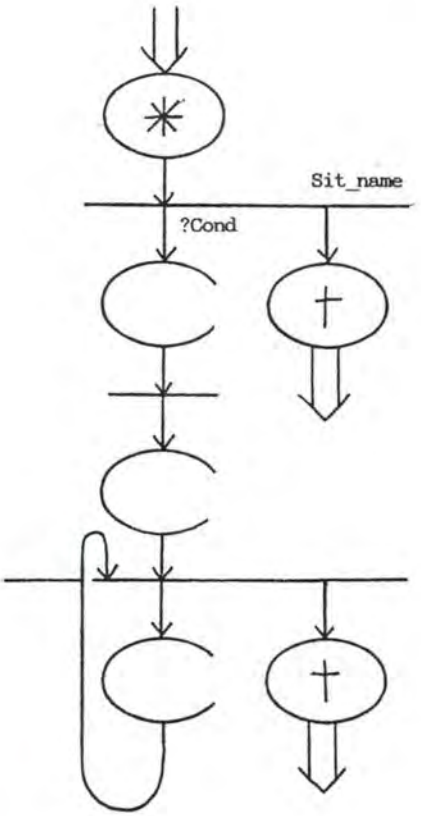
Introduisons maintenant la représentation des différents diagrammes.

Diagramme matriciel :



représente l'objet Obj avec ses attributs Attr1..AttrN et ses événements Evt1..EvtN.

Diagramme de comportement :



o représente le comportement d'une classe d'objet.

o notons que

- - chaque situation peut posséder un nom;
- - il peut y avoir plusieurs événements de mort;
- - chaque saut de situation peut être conditionnel.

Diagramme d'initialisation :

Lorsque survient événement de naissance Evt, les attributs Attr1, Attr2,...,AttrN prennent respectivement les valeurs A, B,...,C

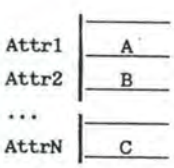
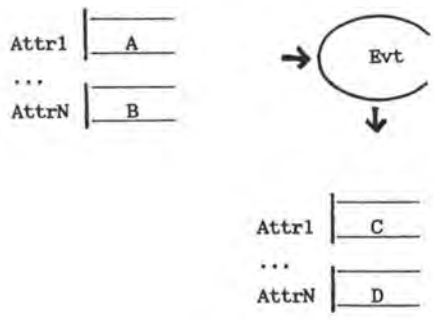


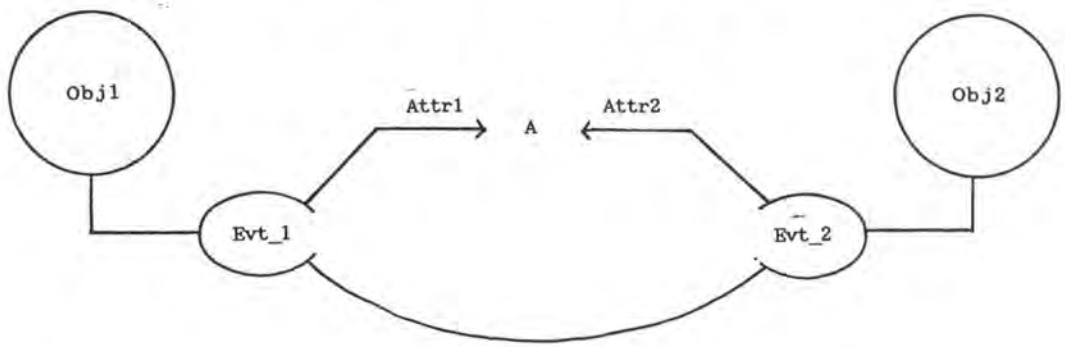
Diagramme de mise à jour :



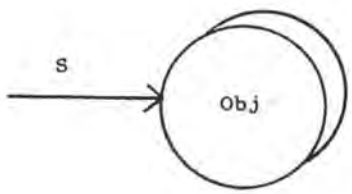
Lorsque survient événement de mise à jour Evt, les attributs Attr1,...,AttrN prennent les valeurs C,...,D. Ces valeurs peuvent être exprimées par des expressions incluant les valeurs A,...,B.

Dans un tel diagramme, il convient de ne détailler que les attributs pertinents.

Diagramme d'interaction et de liaison de valeur



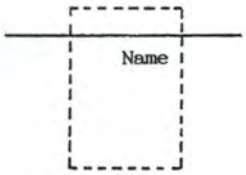
Ce diagramme signifie que lorsque la condition Cond est vérifiée, événement Evt1 de l'objet Obj1 appelle événement Evt2 de l'objet Obj2 et lui passe la valeur A. Celle-ci provient de l'attribut Attr1 de l'objet Obj1 et est transmise à l'attribut Attr2 de l'objet Obj2. Bien entendu, il faut que les types soient compatibles. Notons que la pointe de flèche d'interaction peut être double pour signifier une interaction multiple



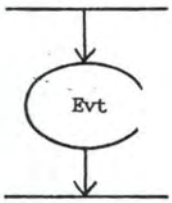
Cette représentation signifie qu'un slot S (qu'il soit attribut d'un objet ou d'un événement) référence une instance précise d'une classe d'objet ouverte.

8.5 Compléments à la notation graphique

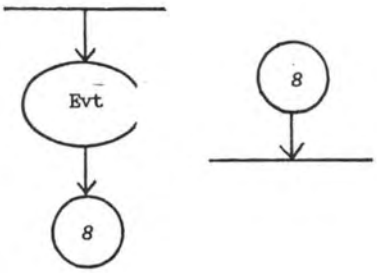
Pour simplifier la lecture des diagrammes de comportement, nous avons introduit deux notations particulières : le module et la numérotation.



représente une partie de schéma déjà décrite ailleurs (entourée également de pointillé et possédant le même nom).



peut être transformé en



Cela permet de découper un schéma de taille importante en sous-schémas.

Troisième Partie

Spécifications et modifications

Dans la première partie, nous avons décrit le comportement de notre prototype. Dans la seconde, nous avons choisi une approche de conception et un langage la supportant. Nous utilisons maintenant ces éléments pour réaliser les spécifications.

Dans un premier temps, nous précisons la méthodologie utilisée pour les réaliser. Le second chapitre contient l'ensemble des spécifications et leur explication. Dans un dernier chapitre, nous montrons comment les adapter en fonction des modifications des besoins.

Chapitre 9

Méthodologie utilisée pour les spécifications

Dans ce chapitre, nous détaillons d'abord le processus de spécification. Ensuite, nous justifions la synthèse que nous avons dû réaliser. Pour terminer, nous expliquons comment nous avons tenté de garantir une indépendance des spécifications par rapport à l'implémentation.

9.1 Processus de spécification

L'équipe qui réalise OBLOG ne présente pas de méthode de développement particulière. Nous avons choisi de considérer les différents schémas comme autant d'étapes différentes et de les aborder dans l'ordre le plus logique possible.

Il nous a semblé que la première étape était de déterminer les objets composant notre application. Les objets sont, comme nous l'avons vu au chapitre 8, définis par un diagramme matriciel. Dans cette première étape, nous avons déterminé le nom de l'objet et les attributs qui s'y rattachent. Ceux-ci représentent en fait les informations d'état que l'on souhaite conserver à propos d'un objet.

D'autres composants des diagrammes matriciels sont les événements. Leur choix peut en fait être considéré comme la deuxième étape dans notre démarche de spécification. Cela revient à se poser la question : "que peut faire cet objet ?" De nouveaux attributs peuvent être découverts lors de cette phase.

Une fois que nous connaissons tous les événements d'un objet, il faut les agencer pour déterminer son diagramme de comportement. On a ainsi une bonne idée des situations par lesquelles passe l'objet au cours de sa vie.

Lorsque les diagrammes de comportements de tous les objets sont réalisés, il convient de définir les diagrammes d'initialisation. Cette étape permet de spécifier les

valeurs des attributs à la naissance de l'objet. Cette connaissance est nécessaire pour passer à la phase suivante.

Celle-ci consiste à réaliser les appels entre événements d'objets distincts. Ces diagrammes d'appel permettent de lier les objets entre eux via leurs événements.

La dernière étape consiste, lorsque cela s'avère nécessaire (c'est-à-dire lorsque les attributs sont modifiés), à décrire les événements de mise-à-jour à l'aide des diagrammes adéquats.

Lorsque tout ces diagrammes sont terminés (et documentés), nous pouvons dire que nous disposons d'une spécification complète de l'application.

9.2 Présentation synthétique

Vu la taille du prototype et la complexité de son comportement, ses spécifications s'étalent sur un nombre impressionnant de pages. Nous avons donc choisi de ne présenter dans cet ouvrage qu'une synthèse des différents schémas.

Les diagrammes matriciels, de comportement et d'initialisation sont présentés in extenso. Il nous paraît en effet assez difficile de les résumer. Notons toutefois l'utilisation de modules et de numéros présentés au point 8.5. Ces mécanismes sont utilisés afin de faciliter la lecture des spécifications.

Les diagrammes de liaison de valeurs et d'interaction ne sont pas tous présentés sous forme de schémas. Si plusieurs diagrammes décrivent une interaction tout à fait semblable, un seul est présenté. Dans les commentaires des différentes interactions, le texte explicatif fait référence à ce schéma unique.

De la même façon, nous n'avons présenté que les diagrammes de mise-à-jour qui nous paraissent apporter une information supplémentaire à celle fournie par les schémas de liaison de valeurs.

Soulignons donc que nous avons tenté au maximum de faciliter la compréhension des spécifications en évitant de noyer le lecteur sous un volume d'informations redondantes.

9.3 Indépendance vis-à-vis de l'implémentation

Nous avons expliqué au chapitre 7 que nous souhaitions des spécifications réutilisables autant que faire ce peut. Il est donc essentiel que celles-ci soient le plus indépendantes possible d'un langage d'implémentation et d'un support physique. Ce souci d'indépendance va apparaître au niveau des entrées et au niveau des sorties.

Commençons par la spécification des entrées et prenons l'exemple du choix de l'option Basic dans l'écran principal. Nous avons désigné cet événement pour l'objet utilisateur par le terme "Ask Basic"[1]. Cela nous permet de rester plus général, et donc plus indépendant de l'implémentation, que si nous avions utilisé "Enter CTRL+B" ou "Click on Basic". De même, le changement de ligne courante dans une liste est appelé "Change CL" (current line) au lieu de "Arrow Up/Down" ou "Click in List".

L'indépendance vis-à-vis de l'implémentation doit également se manifester au niveau des sorties, c'est-à-dire de l'affichage. Poursuivons l'exemple de l'option Basic. Après avoir demandé cette option, l'utilisateur se voit présenter un menu pour choisir une des options d'information de base sur les données principales. Nous avons choisi le terme "Display Basic Menu" pour désigner cet événement. Il ne présage en rien de l'implémentation car il peut autant désigner l'affichage d'un écran (comme dans notre implémentation en PQL) que le déroulement d'un menu. L'affichage d'une barre de menu et des choix qui y sont disponibles a suivi la même règle. Nous utilisons "Display Main Bar" et "Change Basic Disponibility" sans préjuger de la représentation de cette disponibilité. Par exemple dans notre prototype, nous n'affichons pas les options non disponibles alors que certains environnements nous permettraient de les afficher en grisé.

Ce souci de distance par rapport à un système et/ou langage peut poser certains problèmes. Prenons le cas de la sauvegarde d'une zone d'écran. Autant en PQL que dans un environnement Windows il est possible de superposer différentes fenêtres ou boîtes de dialogues tout en conservant la possibilité de récupérer les fenêtres sous-jacentes. Mais cette facilité n'est pas disponible dans tous les systèmes. Pour nos spécifications, nous avons décidé d'adopter une approche hybride : l'affichage et l'effacement d'une boîte de dialogue sont possibles alors que pour un menu ou un écran ils ne le sont pas. Ce choix permet d'illustrer les deux types de spécifications.

---

[1] Notons que les spécifications sont réalisées en anglais afin qu'elles puissent éventuellement être utilisées au CES.

Par exemple, il est possible de faire "Erase Exit Dbox" (Dialog Box) mais "Erase Set-up Screen". Dans ce dernier cas, si le set-up est appelé depuis le menu principal, nous avons les événements "Display Main Screen" et "Display Main CV" (Current Value).

Chapitre 10

Spécifications du prototype

Notre prototype met en présence deux objets fondamentaux : l'utilisateur et l'application. Cette dernière est composée d'une zone de travail, d'une zone de contexte et d'une barre de menu (Voir section 3.1). L'application accède à une base de données et à des fichiers texte. La zone de travail permet d'afficher un texte ou une liste. Celle-ci est composée de lignes. La zone de contexte contient une valeur par défaut et éventuellement une valeur courante de la zone de travail. Celles-ci sont en fait des lignes.

Voici les objets que nous avons choisis à partir de cette description :

- User;
- Working Area;
- Context Area;
- Menu Bar;
- Current Values;
- Default value;
- Database;
- File;
- Text;
- List;
- Line.

10.1 Diagrammes matriciels

10.1.1 L'objet User

L'objet User (Figure 10.1) possède plusieurs attributs :

- filename (string) : désigne le nom du fichier courant auquel l'utilisateur peut accéder;

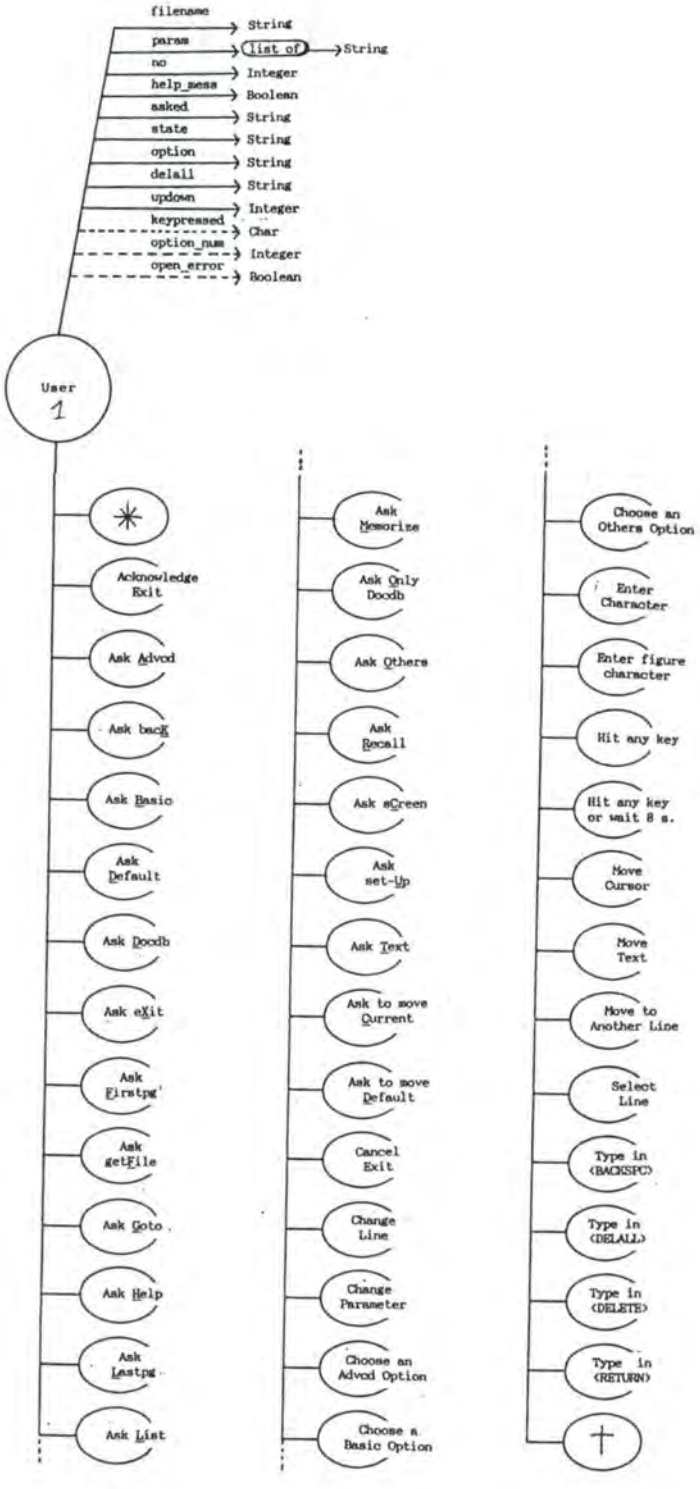

filename
String
parm
list of
String
no
Integer
help_mess
Boolean
asked
String
state
String
option
String
delall
String
updown
Integer
keypressed
Char
option_num
Integer
open_error
Boolean
User
1
Acknowledge Exit
Ask Advcd
Ask backK
Ask Basic
Ask Default
Ask Doodb
Ask eXit
Ask Firstpg
Ask getFile
Ask Goto
Ask Help
Ask Lastpg
Ask List
Ask Memorize
Ask Only Doodb
Ask Others
Ask Recall
Ask sCreen
Ask set-Up
Ask Text
Ask to move Current
Ask to move Default
Cancel Exit
Change Line
Change Parameter
Choose an Advcd Option
Choose a Basic Option
Choose an Others Option
Enter Character
Enter figure character
Hit any key
Hit any key or wait 8 s.
Move Cursor
Move Text
Move to Another Line
Select Line
Type in <BACKSPC>
Type in <DELALL>
Type in <DELETE>
Type in <RETURN>

Figure 10.1

- param1 à 6 (string) : il s'agit des cinq paramètres du menu Set-up et du nom du dernier fichier auquel l'utilisateur a accédé;
- no (integer) : désigne le numéro de ligne auquel l'utilisateur désire accéder dans un texte;
- help_mess (boolean) : est équivalent au troisième paramètre et signale si le message "CTRL+H..." doit apparaître dans le menu initial;
- asked (string) : est équivalent au cinquième paramètre et précise si l'utilisateur désire reprendre le dernier fichier consulté ou décide d'en choisir un;
- state (string) : détermine dans quelle partie du logiciel (Initial, Text ou Main) se trouve l'utilisateur;
- delall (string) : est équivalent au premier paramètre et contient le caractère utilisé pour effacer tous les champs du menu principal;
- updown (string) : est équivalent au quatrième paramètre et spécifie le nombre de lignes à faire défiler avec les flèches dans le menu Texte;
- keypressed (char, partiel) : détermine quelle touche a été entrée par l'utilisateur;
- option_num (integer, partiel) : désigne le numéro de l'option dans un menu;
- open-error (boolean, partiel) : permet de signaler à l'utilisateur que l'ouverture d'un fichier s'est déroulée correctement.

Outre sa naissance et sa mort, plusieurs événements peuvent se produire dans l'objet User :

- Acknowledge Exit : l'utilisateur confirme son désir de sortir;
- Ask *X* : l'utilisateur désire accéder à l'option *X*. Les différentes options existantes sont Advcd, bacK, Basic, Default, Docdb, eXit, Firstpg, getFile, Goto, Help, Lastpg, List, Memorize, Only Docdb, Others, Recall, sCreen, set-Up, Text, to move Current, to move Default;
- Cancel Exit : l'utilisateur renonce à quitter l'application;
- Change Line : l'utilisateur désire changer de ligne dans une liste;
- Change Parameter : l'utilisateur souhaite modifier le paramètre courant dans le menu Set-up;
- Choose an *X* Option : l'utilisateur choisit une option dans le menu *X*. Les menus sont Advcd, Basic et Others;
- Enter character et Enter figure character : l'utilisateur entre un caractère respectivement alphanumérique et numérique;
- Hit any key : l'utilisateur frappe n'importe quelle touche;

- Hit any key or wait 8 s. : dans le cas d'un message d'erreur, l'utilisateur frappe n'importe quelle touche pour le faire disparaître ou attend huit secondes qu'il s'efface automatiquement;
- Move Cursor, Move Text et Move to Another Line : l'utilisateur désire respectivement déplacer le curseur, le texte ou aller vers une autre ligne du Set-up. Rappelons que ces termes sont choisis de façon à ne pas présager de l'implémentation;
- Type in <*X*> : l'utilisateur frappe la touche *X* où *X* est BACKSPC, DELALL, DELETE ou RETURN;

10.1.2 L'objet Working Area

L'objet Working Area (Figure 10.2) possède plusieurs attributs :

- help_mess (boolean) : signale si le message "CTRL+H..." doit apparaître dans le menu initial;
- ask_for_file (string) : précise si l'utilisateur désire reprendre le dernier fichier consulté ou décide d'en choisir un;
- default (objet Line) : désigne la valeur par défaut;
- cur_line (objet Line) : représente la valeur courante;
- n° (integer) : désigne un numéro de ligne utilisé lors des consultations de liste.
- current_list (objet List) : représente la liste affichée dans l'écran courant.

Outre sa naissance et sa mort, plusieurs événements peuvent se produire dans l'objet Working Area :

- Change CL (Current Line) : change la ligne courante dans la liste affichée à l'écran. Rappelons qu'une ligne courante est contrastée à l'écran;
- Change CP (Current Parameter) : change la valeur du paramètre courant
- Change CV (Current Values) : change la valeur courante;
- Change Set-up CL : change la ligne courante dans le Set-up;
- Display *X* CL : change la ligne courante dans un menu ou une liste. *X* peut être Advcd, Basic, Goto et Others pour les menus et Byyear, Freq, Path, Posi, Record pour les listes;
- Display *X* Content : affiche le contenu de l'écran *X*. *X* est un des écrans suivants : Advdst, Advvar, Basdst, Bassty, Bassvy, Basvar;
- Display *X* Screen : affiche l'écran vide. *X* peut être Advdst, Advvar, Basdst, Bassty, Bassvy, Basvar, Byyear, Freq, Initial, Main, Memorized, Path, Posi, Record, Set-up, Text;

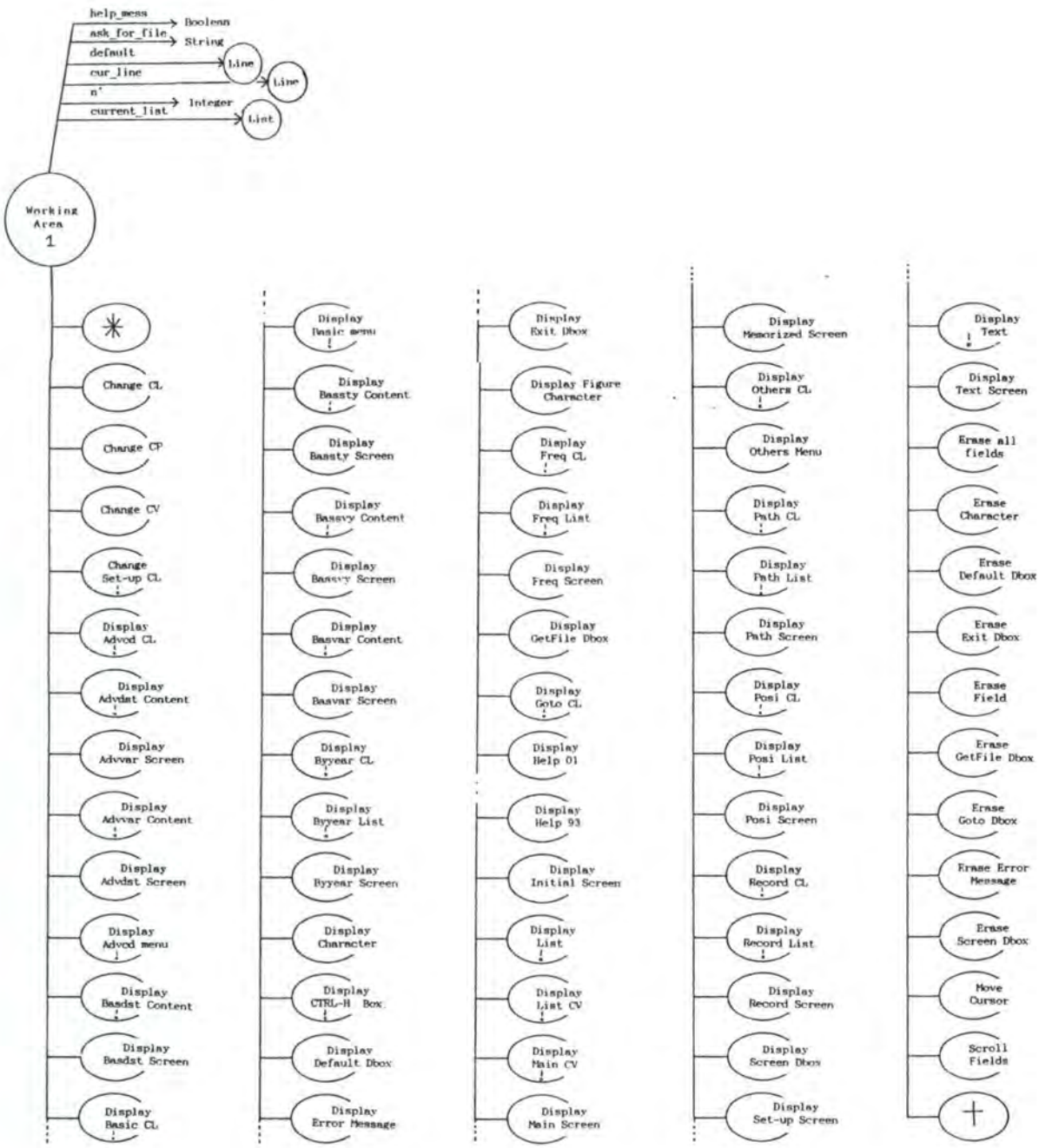

help_mess
Boolean
ask_for_file
String
default
Line
cur_line
Line
n°
Integer
current_list
List
Working Area 1
Change CL
Change CP
Change CV
Change Set-up CL
Display Advcd CL
Display Advdst Content
Display Advvar Screen
Display Advvar Content
Display Advdst Screen
Display Advcd menu
Display Basdst Content
Display Basdst Screen
Display Basic CL
Display Basic menu
Display Bassty Content
Display Bassty Screen
Display Bassvy Content
Display Bassvy Screen
Display Basvar Content
Display Basvar Screen
Display Byyear CL
Display Byyear List
Display Byyear Screen
Display Character
Display CTRL-H Box
Display Default Dbox
Display Error Message
Display Exit Dbox
Display Figure Character
Display Freq CL
Display Freq List
Display Freq Screen
Display GetFile Dbox
Display Goto CL
Display Help 01
Display Help 93
Display Initial Screen
Display List
Display List CV
Display Main CV
Display Main Screen
Display Memorized Screen
Display Others CL
Display Others Menu
Display Path CL
Display Path List
Display Path Screen
Display Posi CL
Display Posi List
Display Posi Screen
Display Record CL
Display Record List
Display Record Screen
Display Screen Dbox
Display Set-up Screen
Display Text
Display Text Screen
Erase all fields
Erase Character
Erase Default Dbox
Erase Exit Dbox
Erase Field
Erase GetFile Dbox
Erase Goto Dbox
Erase Error Message
Erase Screen Dbox
Move Cursor
Scroll Fields

Figure 10.2

- Display *X* Menu : affiche un menu. *X* est soit Advcd, soit Basic, soit enfin Others.
- Display *X* List : affiche une liste. *X* peut être Byyear, Freq, Path, Posi, Record;
- Display Character et Display Figure Character : affiche un caractère respectivement alphanumérique et numérique;
- Display CTRL-H Box : affiche dans le menu initial le message expliquant à l'utilisateur comment accéder à l'aide;
- Display *X* Dbox (Dialog Box) : affiche une boîte de dialogue pour l'option *X*. *X* est Default, Exit, Getfile, Goto, Screen;
- Display Error Message : affiche un message d'erreur et son numéro dans une boîte de dialogue;
- Display Help *X* : affiche l'écran d'aide numéro *X*;
- Display List : affiche la liste dans l'option List:
- Display List CV : affiche la valeur courante de cette liste;
- Display Main CV : affiche la valeur courante dans le menu principal;
- Display Text : affiche le texte dans l'option Texte;
- Erase Character, Field, all fields : efface respectivement un caractère, le contenu d'un champ ou le contenu de tous les champs du menu principal;
- Erase X Dbox : efface la boîte de dialogue de l'option X (voir 9.3). X est une des options suivantes : Default, Exit, Getfile, Goto, Screen;
- Erase Error Message : efface la boîte de dialogue du message d'erreur;
- Move Cursor : déplace le curseur dans les champs du menu principal;
- Scroll Fields : déroule les champs du menu principal pour l'option List.

10.1.3 L'objet Context Area

L'objet Context Area (Figure 10.3) possède plusieurs attributs :

- char (char) : désigne le caractère entré dans la zone de contexte dans l'option GetFile;
- current (objet Line) : représente la ligne de valeur courante affichée;
- name (string) : précise le nom du fichier dans l'option texte;
- default (objet Line) : représente la ligne de valeur par défaut affichée;
- curpos (integer) : désigne la position du curseur dans le nom de fichier (pour l'option GetFile).

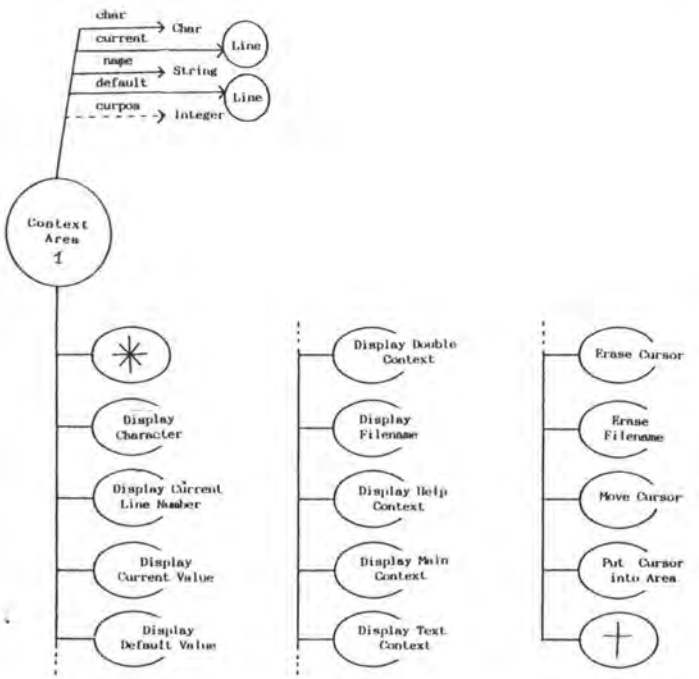


Figure 10.3

Outre sa naissance et sa mort, plusieurs événements peuvent se produire dans l'objet Context Area :

- Display Character : affiche un caractère;
- Display Current Line Number : affiche le numéro de ligne courante du texte;
- Display Current Values et Display Default Value : affiche respectivement les valeurs courantes et par défaut;
- Display *X* Context : affiche le contexte *X*. *X* peut être Double, Help, Main ou Text;
- Display Filename : affiche le nom du fichier dans l'option texte;
- Erase Cursor : efface le curseur;
- Move Cursor : déplace le curseur;
- Put Cursor into Area : affiche le curseur en début de zone.

10.1.4 L'objet Menu Bar

L'objet Menu Bar (Figure 10.4) possède plusieurs attributs booléens qui déterminent la disponibilité des différentes options reprises dans les barres de menu. Celles-ci sont : exit, help, set-up, back, firstpg, lastpg, docdb, getfile, goto, screen, list, basic, others, advcd et default.

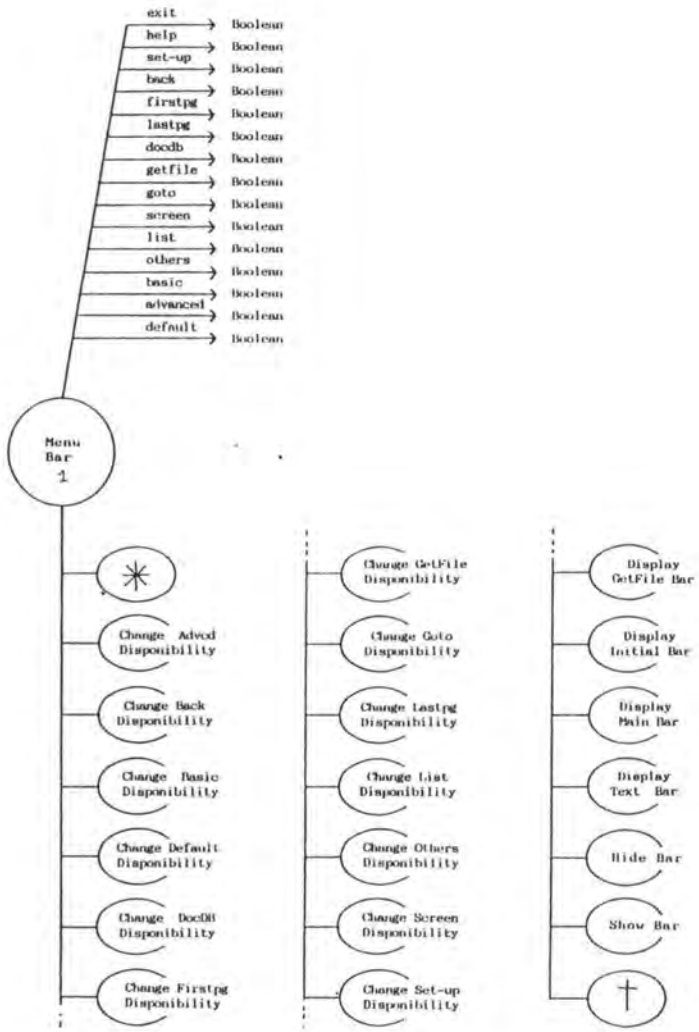


Figure 10.4

Outre sa naissance et sa mort, plusieurs événements peuvent se produire dans l'objet Menu Bar:

- Change *X* Disponibility : change la disponibilité (et son affichage) de l' option *X*. *X* peut être Advcd, Back, Basic, Default, Docdb, Firstpg, Getfile, Goto, Lastpg, List, Others, Screen ou Set-up;
- Display *X* Bar : affiche la barre de menu pour l'option *X*. *X* est GetFile, Initial, Main ou Text;
- Hide Bar et Show Bar permettent respectivement de cacher et de réafficher la barre de menu. Notons que, comme pour la boîte de dialogue, nous avons considéré que cette dernière opération est possible. Sinon, elle peut être simulée par l'affichage de la barre de menu originelle et les changements de disponibilité adéquats.

10.1.5 L'objet Current Values

L'objet Current Values (Figure 10.5) possède plusieurs attributs :

- num (integer) : représente le numéro de la ligne à lire dans une liste;
- type (string) : désigne le type de la ligne à lire;
- top (objet Line) : représente la ligne à lire ou à transférer;
- current_lines (pile d'objet Line) : représente la pile de valeurs courantes. Rappelons que l'appel d'options imbriquées a pour conséquence que chaque niveau a une valeur courante différente. Le sommet de la pile contient la valeur courante dans l'option courante.

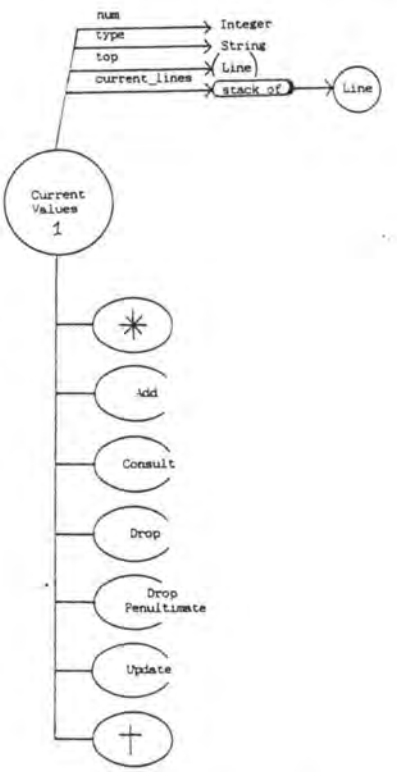


Figure 10.5

Outre sa naissance et sa mort, plusieurs événements peuvent se produire dans l'objet Current Values :

- Add : ajoute une ligne sur la pile;
- Consult : permet de consulter une ligne de la pile;
- Drop : supprime la ligne au sommet de la pile;

- Drop Penultimate : supprime l'avant-dernier élément de la pile. Notons que cela peut être implémenté par deux Drop suivis d'un Add de la première valeur supprimée;
- Update : met à jour la ligne au sommet de la pile.

10.1.6 L'objet Default Value

L'objet Default Value (Figure 10.6) ne possède qu'un attribut default_line (objet Line) qui représente la ligne à lire ou à transférer.

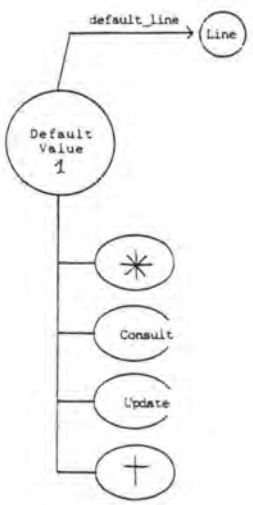


Figure 10.6

Outre sa naissance et sa mort, plusieurs événements peuvent se produire dans l'objet Default Value :

- Consult : permet de consulter la valeur;
- Update : met à jour cette même valeur.

10.1.7 L'objet Database

L'objet Database (Figure 10.7) possède plusieurs attributs :

- num (integer) : précise le numéro de la fonction de l'option List à exécuter;
- function (string) : désigne la fonction d'accès exécutée. La valeur de cet attribut détermine à quel événement la liste résultat est passée;
- list (objet List) : représente la liste résultat de l'accès;

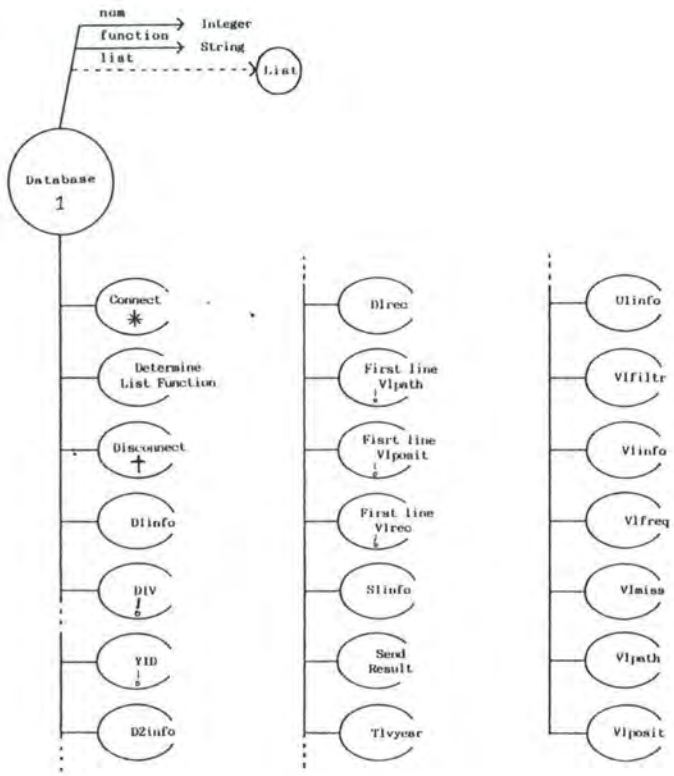


Figure 10.7

Plusieurs événements peuvent se produire dans l'objet Database :

- Connect et Disconnect : il s'agit respectivement des événements de naissance et de mort de l'objet Database. La mort ne signifie pas que la base de données est détruite mais que l'application ne peut plus y accéder;
- Determine List Function : détermine quelle fonction d'accès doit être employée dans l'option List;
- D1V...Y1D : ce sont les fonctions d'accès pour l'option List (voir annexe 5);
- D1info, D2info, Dlrec, S1info, Tlvyear, U1info, V1filtr, V1info, Vlfreq, Vlmiss, Vlpath, Vlposit : il s'agit des fonctions d'accès pour les options Advcd, Basic et Others (pour plus de détails sur celles-ci, voir point 2.5.3);
- First Line Vlpath, First Line Vlposit et First Line Vlrec : fournissent respectivement la première ligne des fonctions Vlpath, Vlposit, Vlrec;
- Send Result : renvoit la liste résultat.

10.1.8 L'objet File

L'objet File (Figure 10.8) possède plusieurs attributs :

- open_error (boolean) : précise si l'opération d'ouverture du fichier s'est bien déroulée;
- name (string) : représente le nom du fichier texte;
- file_text (liste de string) : contient les lignes du fichier texte.

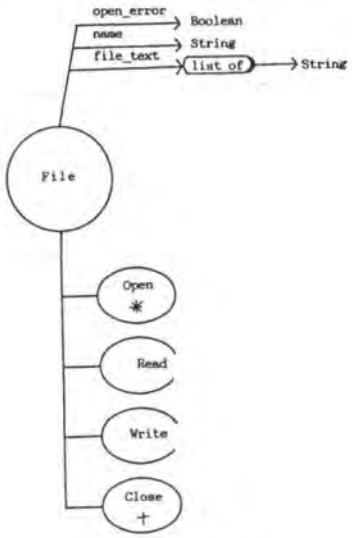


Figure 10.8

Plusieurs événements peuvent se produire dans l'objet File :

- Open et Close : il s'agit respectivement des événements de naissance et de mort de l'objet File;
- Read : permet de lire le contenu du fichier;
- Write : permet de l'écrire.

10.1.9 L'objet Text

L'objet Text (Figure 10.9) possède plusieurs attributs :

- text_lines (liste de string) : contient le texte;
- num (integer) : désigne le numéro de la ligne consultée;
- tline (string) : représente une ligne de texte.

Outre sa naissance et sa mort, plusieurs événements peuvent se produire dans l'objet Text :

- Consult : permet de consulter une ligne de texte étant donné son numéro;
- Fill in : remplit la liste de lignes de texte.

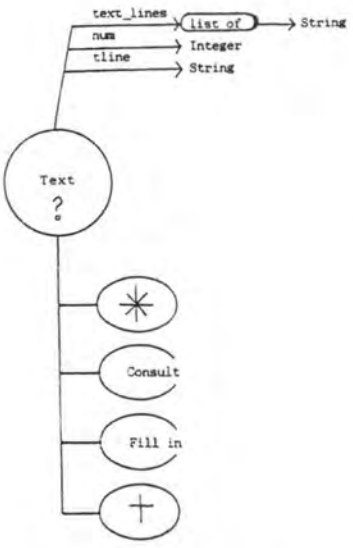


Figure 10.9

10.1.10 L'objet List

L'objet List (Figure 10.10) possède plusieurs attributs :

- line (objet Line) : représente une ligne de la liste;
- num (integer) : détermine le numéro de la ligne à laquelle on désire accéder;
- line_list (liste d'objets Line) : désigne la liste proprement dite;
- type (string) : représente le type de ligne dont la liste est composée.

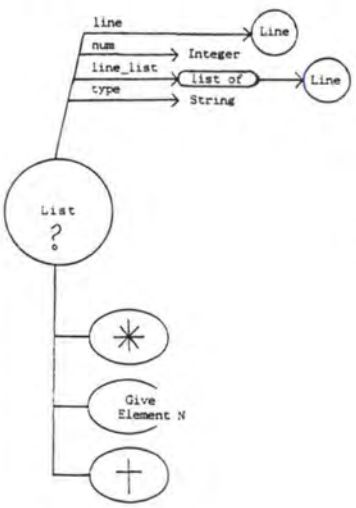


figure 10.10

Outre sa naissance et sa mort, un seul événement peut se produire dans l'objet List : Give Element N, permet de consulter une ligne de numéro donné.

10.1.11 Les objets Line

L'objet Line (Figure 10.11) est un objet incorporateur. Il possède un seul attribut, type (string), qui détermine le type de ligne incorporée. Les objets composants sont Info Line, Basvar Line, Basdst Line, Advdst Line, Quest Line, Missing Line, Record Line, Filter Line, Survey Line, Study Line, Others Line et Frequencies Line.

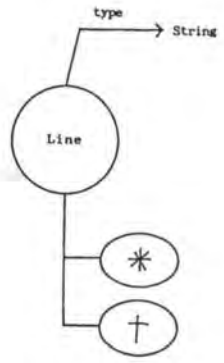


Figure10.11

Line ne possède que les événements de naissance et de mort : les autres événements définissant son comportement sont ceux des objets composants. La figure 10.12 (page suivante) montre cette incorporation et ses conditions.

Les objets composants ont pour attributs les paramètres des fonctions d'accès (voir point 2.5.3). Détaillons à titre d'exemple l'objet Info Line (Figure 10.13).

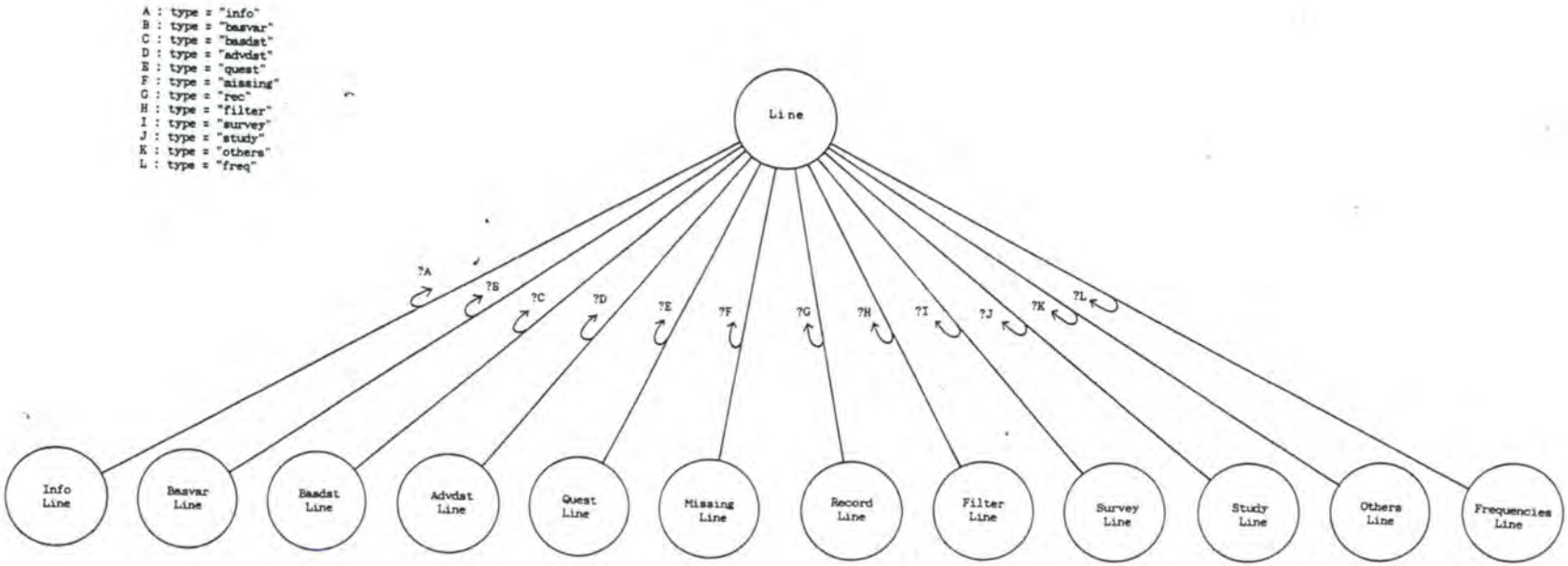

A : type = "info"
B : type = "basvar"
C : type = "basdst"
D : type = "advdst"
E : type = "quest"
F : type = "missing"
G : type = "rec"
H : type = "filter"
I : type = "survey"
J : type = "study"
K : type = "others"
L : type = "freq"
Line
?A
?B
?C
?D
?E
?F
?G
?H
?I
?J
?K
?L
Info Line
Basvar Line
Basdst Line
Advdst Line
Quest Line
Missing Line
Record Line
Filter Line
Survey Line
Study Line
Others Line
Frequencies Line

Figure 10.12

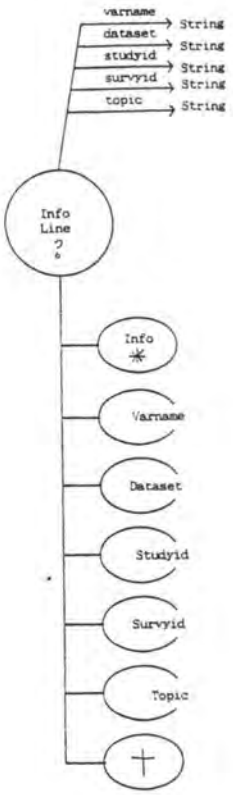
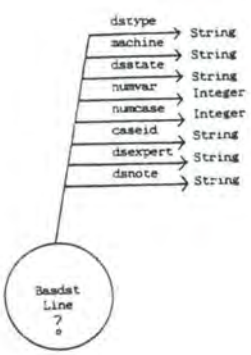
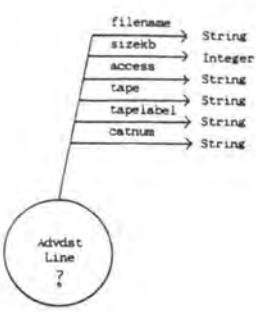
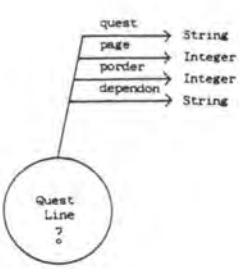


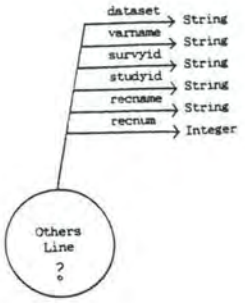
Figure 10.13

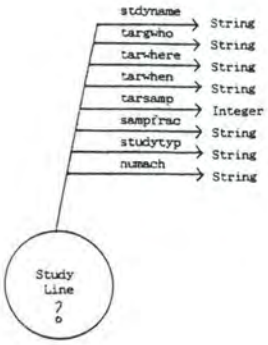
Il possède cinq attributs : varname, dataset, study, survey et topic (tous de type string). Outre ses événements de naissance et de mort, il n'en possède que cinq qui permettent la consultation de ses attributs.

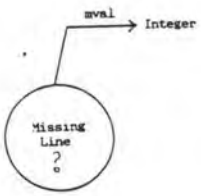


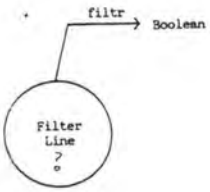


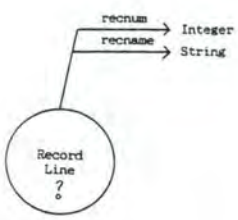


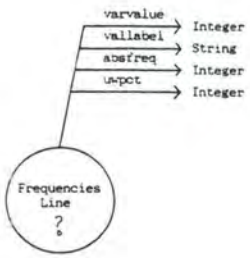

dataset
String
varname
String
survyid
String
studyid
String
recname
String
recnum
Integer
Others
Line
?
°

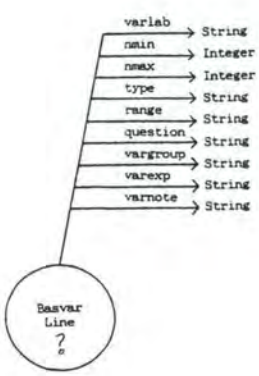

stdyname
String
targ-who
String
tar-here
String
tar-when
String
tarsamp
Integer
sampfrac
String
studytyp
String
numach
String
Study
Line
?
°


mval
Integer
Missing
Line
?
°


filtr
Boolean
Filter
Line
?
°


recnum
Integer
recname
String
Record
Line
?
°


varvalue
Integer
vallabel
String
absfreq
Integer
w-pct
Integer
Frequencies
Line
?
°


varlab
String
nmin
Integer
nmax
Integer
type
String
range
String
question
String
vargroup
String
varexp
String
varnote
String
Basvar
Line
?
°

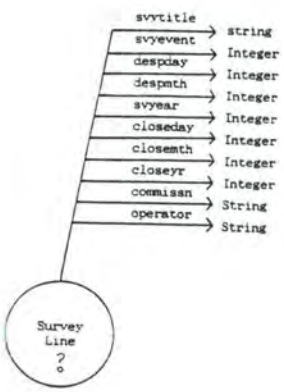

svytitle
svyevent
string
despday
Integer
despmth
Integer
svyear
Integer
closeday
Integer
closemth
Integer
closeyr
Integer
commissn
Integer
operator
String
String
Survey
Line
?
°

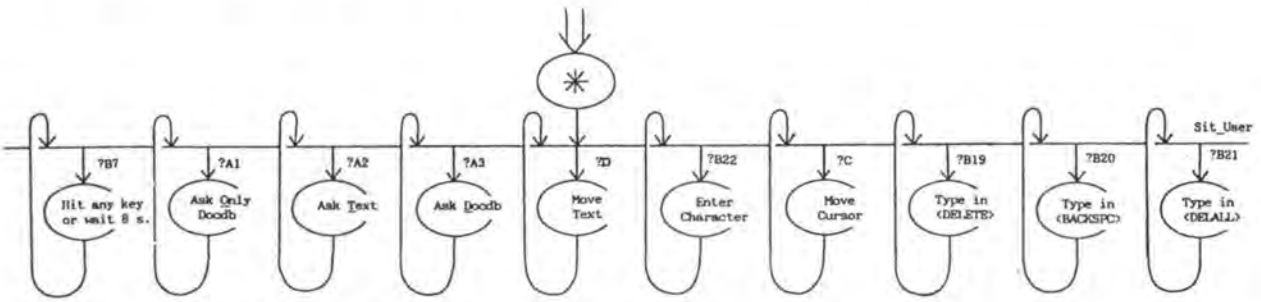
Figure 10.14

La figure 10.14 présente les attributs des autres lignes.

10.2 Diagrammes de comportement

10.2.1 Comportement de l'objet User

Ce diagramme présente l'enchaînement des actions que l'utilisateur peut entreprendre. Notons que ce diagramme utilise les mécanismes de modules et de numérotation décrits au point 8.5.




```
A1  : (state="initial")∧(Only Docdb has been asked)
A2  : (state="initial")∧(Text option has been asked)
A3  : (state="text")∧(Docdb has been asked)
B7  : (A key has been hit or 8 seconds passed)
B19 : (state="main")∧(<DEL> has been type in)
B20 : (state="main")∧(<BACKSPC> has been type in)
B21 : (state="main")∧(<DELALL> has been type in)
B22 : (state="main")∧(An alphanumeric character has been typed
      in)
C   : (Cursor moves, whatever the implementation)
D   : (Move text, whatever the implementation)
```

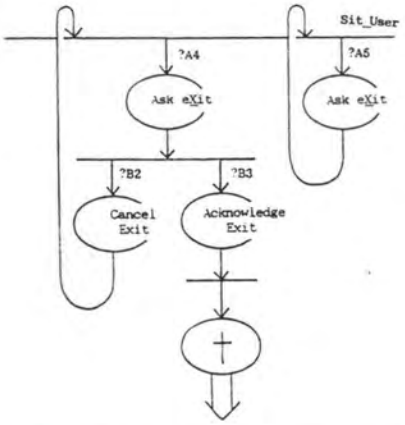
Figure 10.15

La figure 10.15 présente la naissance de l'objet User et les actions les plus simples qu'il peut poser. Remarquons que toute sous-tâche terminée ramène l'utilisateur dans la situation initiale (dans la figure 10.15, les sous-tâches sont des événements atomiques).

La figure 10.16 représente l'option Exit. Notons que les conditions A4 et A5 déterminent quel effet a la demande de sortie (retour au texte ou boîte de dialogue pour la confirmation de sortie). Cette figure est désignée plus loin comme le module Exit.

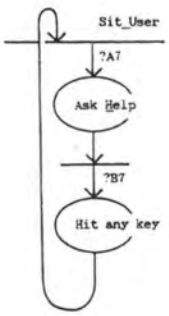
La figure 10.17 présente l'option Help. Cette figure est désignée plus loin comme le module Help.

La figure 10.18 représente l'option Set-up. Cette figure est désignée plus loin comme le module Set-up.



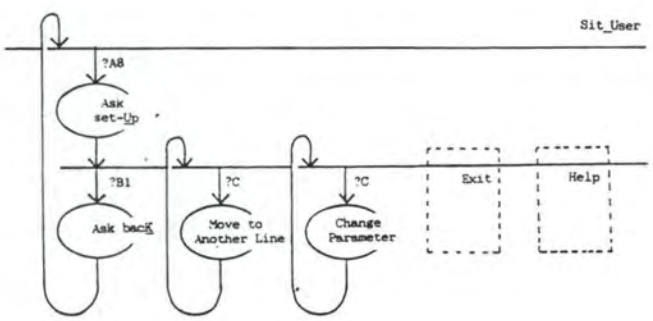
A4 : ((state="text")∧(eXit has been asked))∨
((state="main")∧(option="main")∧(eXit has been asked))
A5 : (state="main")∧(option="text")∧(eXit has been asked)
B2 : (No has been choosen)
B3 : (Yes has been asked)

Figure 10.16



A7 : (Help has been asked)
B7 : (A key has been hit or 8 seconds passed)

Figure 10.17



A8 : (set-Up has been asked)
B1 : (backS has been asked)
C : (Cursor moves, whatever the implementation)

Figure 10.18

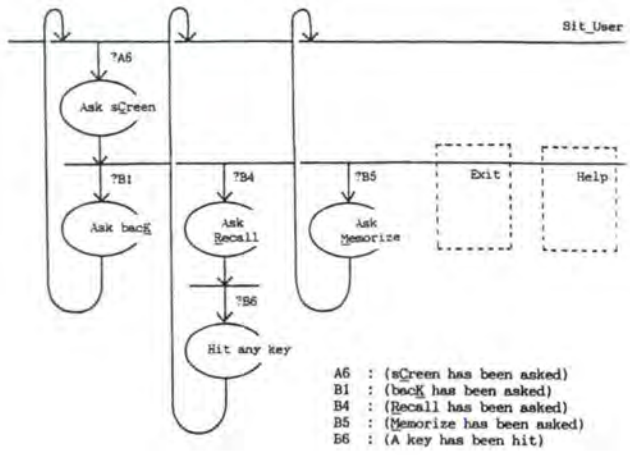


Figure 10.19

La figure 10.19 présente l'option Screen. Cette figure est désignée plus loin comme le module Screen.

La figure 10.20 représente l'option Basic. Elle contient trois sous-tâches. De gauche à droite sont détaillées la demande de Basic alors que la fonction n'est pas autorisée, la demande de Basic alors qu'une seule option est disponible et enfin la demande de Basic qui aboutit au Menu Basic. Notons l'emploi des numéros pour se retrouver dans une situation équivalente dans l'option Advanced (voir Figure 10.21). La figure 10.20 est désignée plus loin comme module Basic.

La figure 10.21 présente l'option Advanced. Elle contient également trois sous-tâches, identiques à celles de Basic. Les numéros renvoient à une situation équivalente dans l'option Basic (voir Figure 10.20). La figure 10.21 est désignée plus loin comme module Advanced.

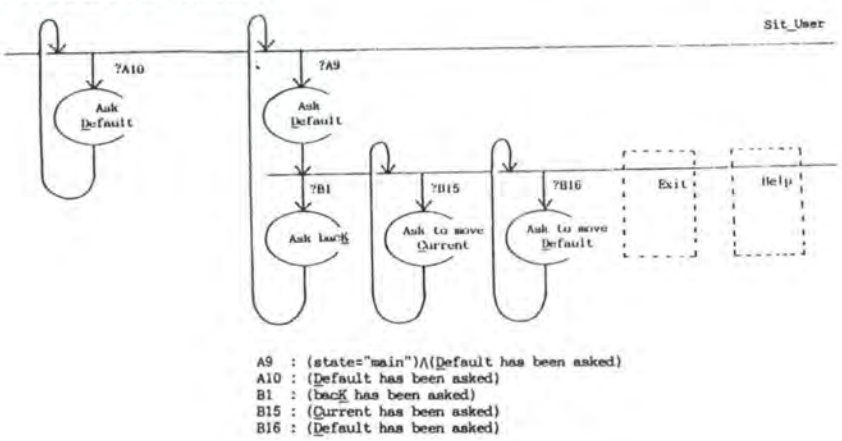
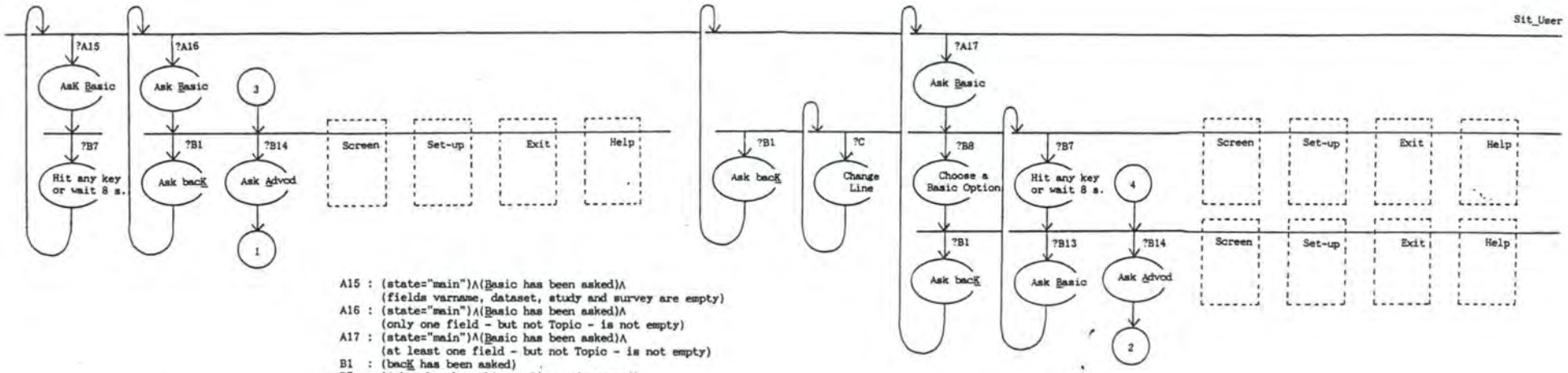


Figure 10.22



A15 : (state="main")∧(Basic has been asked)∧
(fields varname, dataset, study and survey are empty)
A16 : (state="main")∧(Basic has been asked)∧
(only one field - but not Topic - is not empty)
A17 : (state="main")∧(Basic has been asked)∧
(at least one field - but not Topic - is not empty)
B1 : (backK has been asked)
B7 : (A key has been hit or 8 seconds passed)
B8 : (A line has been validated)
B13 : (Basic has been asked)
B14 : (Advod has been asked)
C : (Cursor moves, whatever the implementation)

Figure 10.20

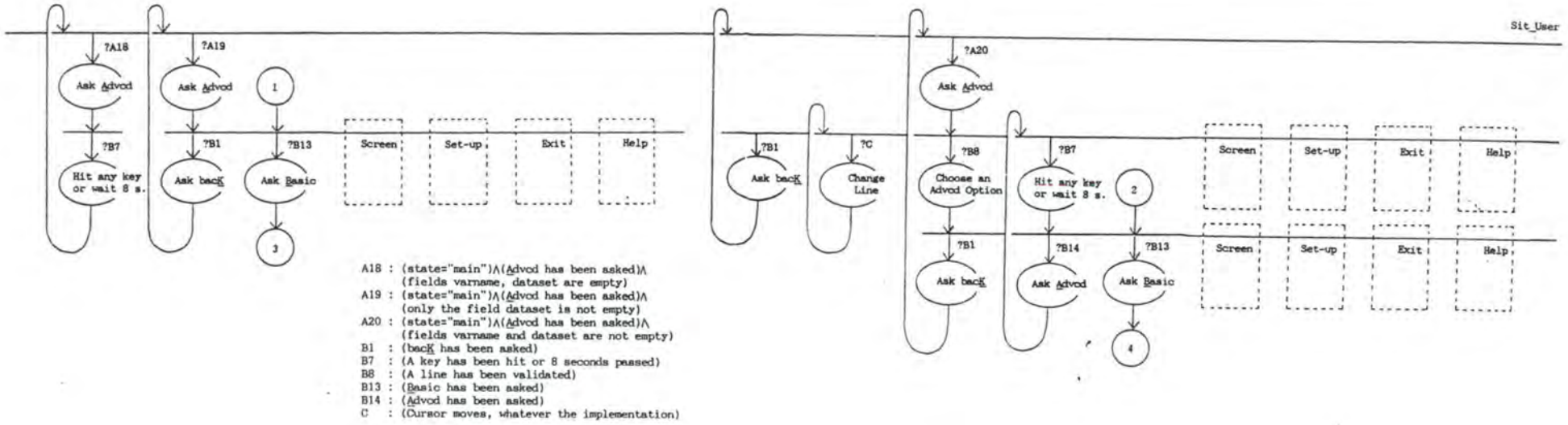
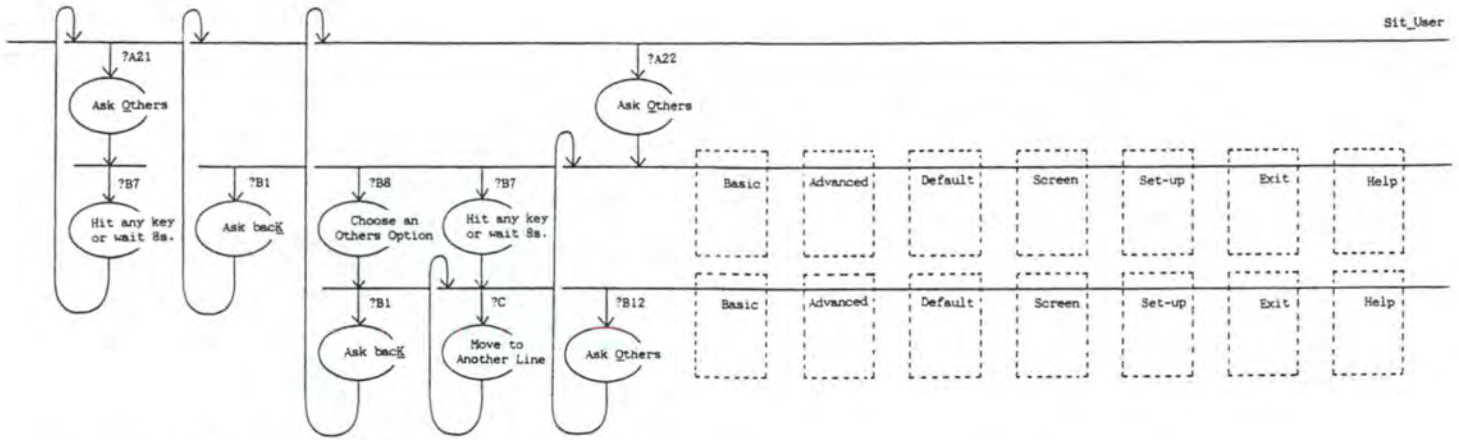

Sit_User
?A18
Ask Advod
?B7
Hit any key or wait 8 s.
?A19
Ask Advod
?B1
Ask backK
1
?B13
Ask Basic
3
Screen
Set-up
Exit
Help
?B1
Ask backK
?C
Change Line
?A20
Ask Advod
?B8
Choose an Advod Option
?B1
Ask backK
?B7
Hit any key or wait 8 s.
2
?B14
Ask Advod
?B13
Ask Basic
4
Screen
Set-up
Exit
Help
Screen
Set-up
Exit
Help
A18 : (state="main")∧(Advod has been asked)∧
(fields varname, dataset are empty)
A19 : (state="main")∧(Advod has been asked)∧
(only the field dataset is not empty)
A20 : (state="main")∧(Advod has been asked)∧
(fields varname and dataset are not empty)
B1 : (backK has been asked)
B7 : (A key has been hit or 8 seconds passed)
B8 : (A line has been validated)
B13 : (Basic has been asked)
B14 : (Advod has been asked)
C : (Cursor moves, whatever the implementation)

Figure 10.21



A21 : (state="main")∧(Others has been asked)∧
(all fields are empty)
A22 : (state="main")∧(Others has been asked)∧
(at least one of the varname, dataset or topic field
is not empty)
B1 : (backK has been asked)
B2 : (No has been choosen)
B7 : (A key has been hit or 8 seconds passed)
B8 : (A line has been validated)
B12 : (Others has been asked)
C : (Cursor moves, whatever the implementation)

Figure 10.24

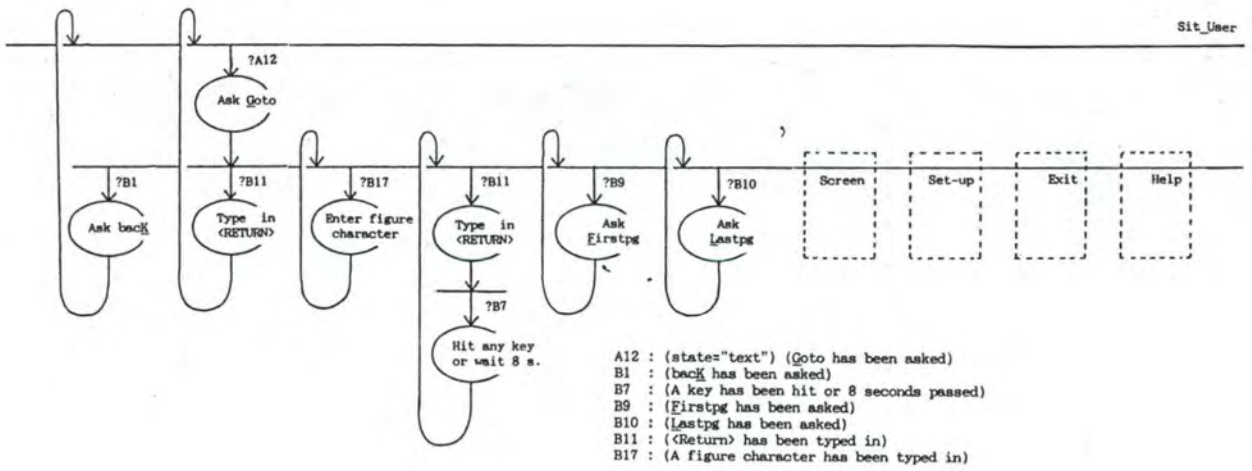

Sit_User
?A12
Ask Goto
?B1
Ask backK
?B11
Type in <RETURN>
?B17
Enter figure character
?B11
Type in <RETURN>
?B7
Hit any key or wait 8 s.
?B9
Ask Firstpg
?B10
Ask Lastpg
Screen
Set-up
Exit
Help
A12 : (state="text") (Goto has been asked)
B1 : (backK has been asked)
B7 : (A key has been hit or 8 seconds passed)
B9 : (Firstpg has been asked)
B10 : (Lastpg has been asked)
B11 : (<Return> has been typed in)
B17 : (A figure character has been typed in)

Figure 10.25

La figure 10.22 représente l'option Default. Rappelons que le comportement de cette option est différent selon que l'on se trouve ou non dans le menu principal. Cette figure est désignée plus loin comme le module Default.

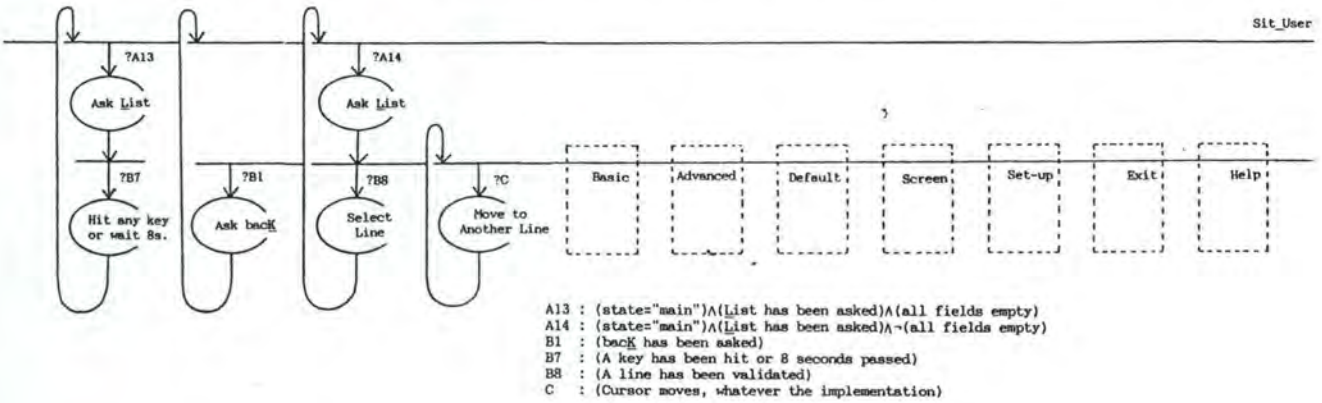


Figure 10.23

La figure 10.23 présente l'option List. Comme les options Basic et Advanced, il est possible qu'une erreur survienne après la demande de l'option si cette demande est inappropriée.

Il en va de même pour l'option Others représentée à la figure 10.24.

La figure 10.25 présente l'option Goto.

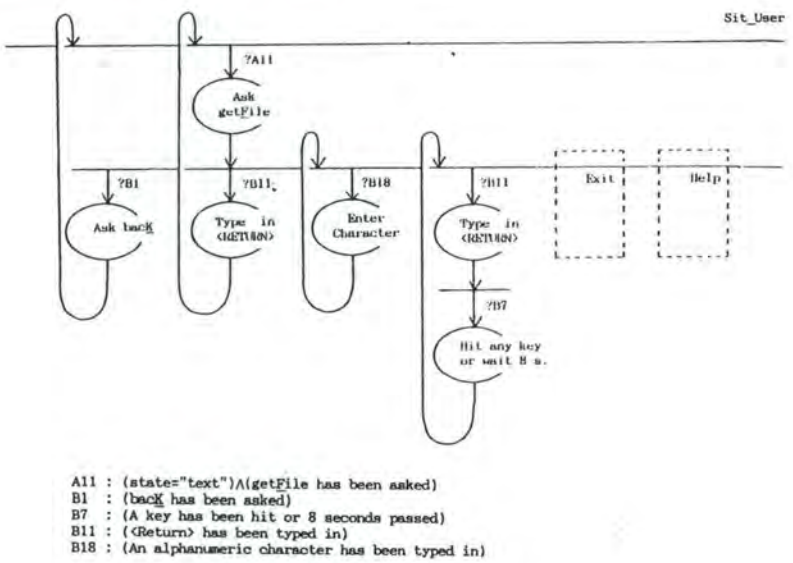


Figure 10.26

Pour terminer, la figure 10.26 représente l'option Getfile.

10.2.2 Comportement de l'objet Working Area

Ce diagramme présente l'enchaînement des événements dans la zone de travail. Notons que ce diagramme utilise le mécanisme de numérotation décrit au point 8.5. Remarquons que chaque situation (exceptée la première) correspond à un écran différent.

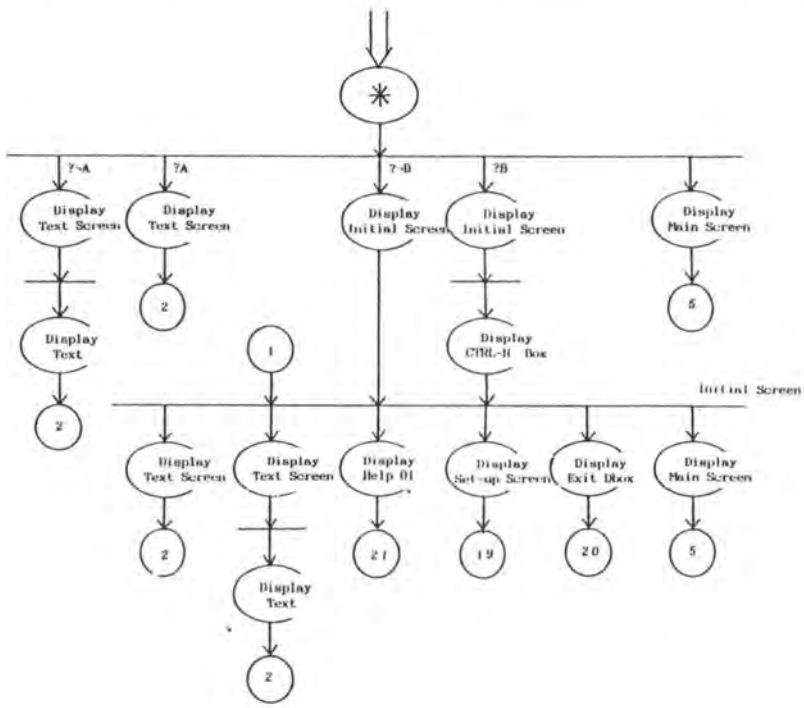


Figure 10.27

La figure 10.27 présente la naissance de l'objet Working Area. Rappelons que le premier écran affiché peut être soit l'écran initial, soit l'écran texte avec le dernier texte utilisé lors de la précédente session, soit ce même écran sans texte, soit enfin l'écran principal. Les conditions de cette figure sont :

A : (ask_for_file = "be asked")

B : (help_mess = True)

La figure 10.28 représente l'écran Texte et ses options GetFile et Goto.

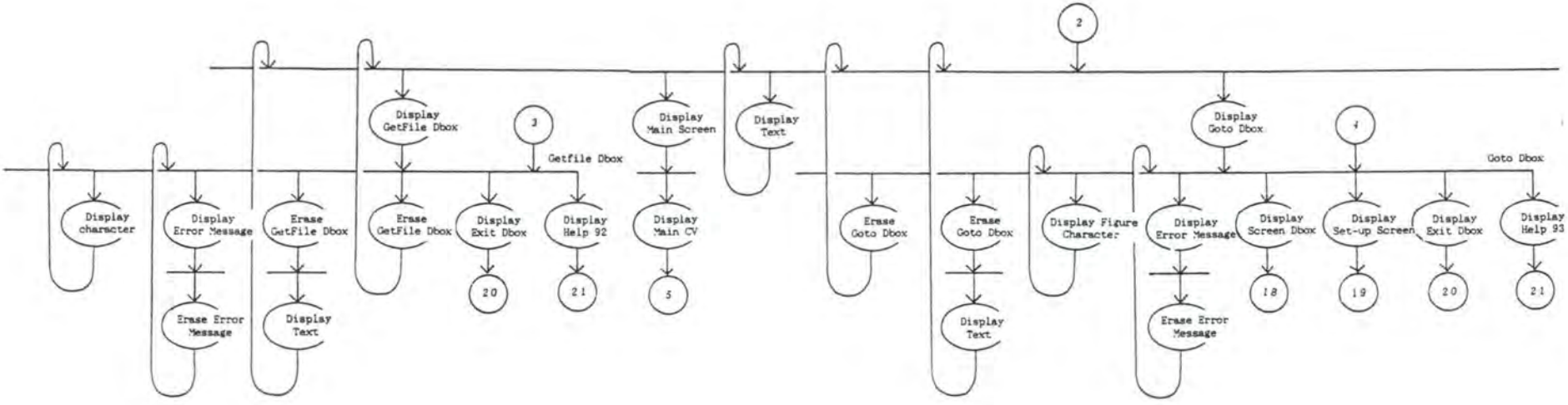


Figure 10.28 (I)

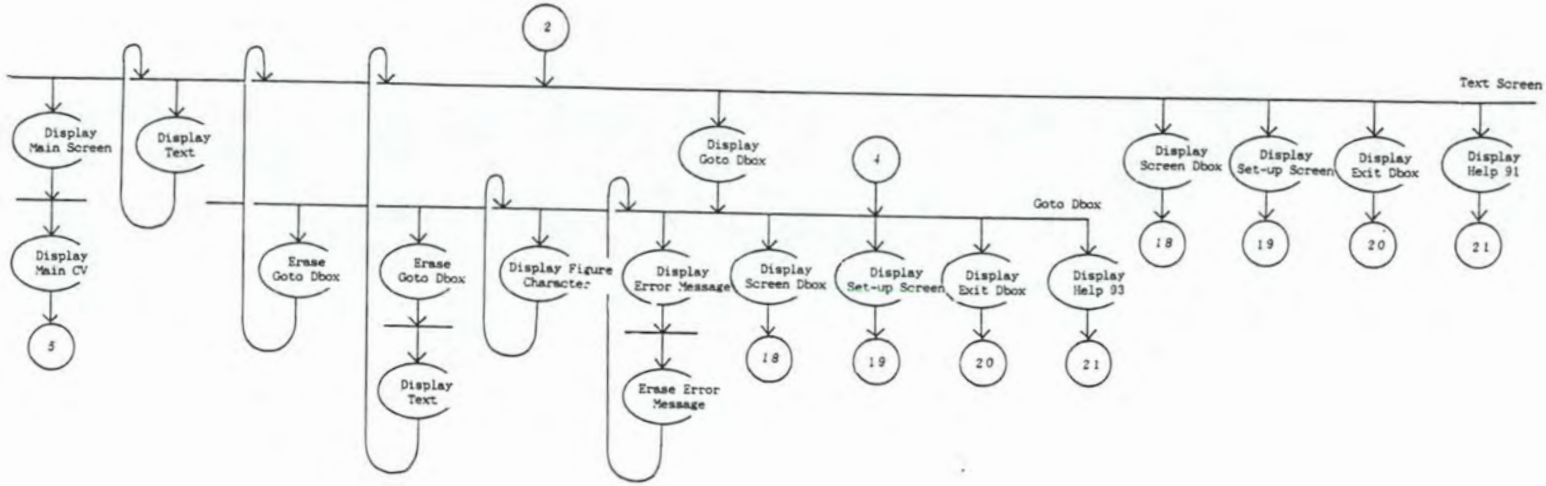


Figure 10.28 (II)

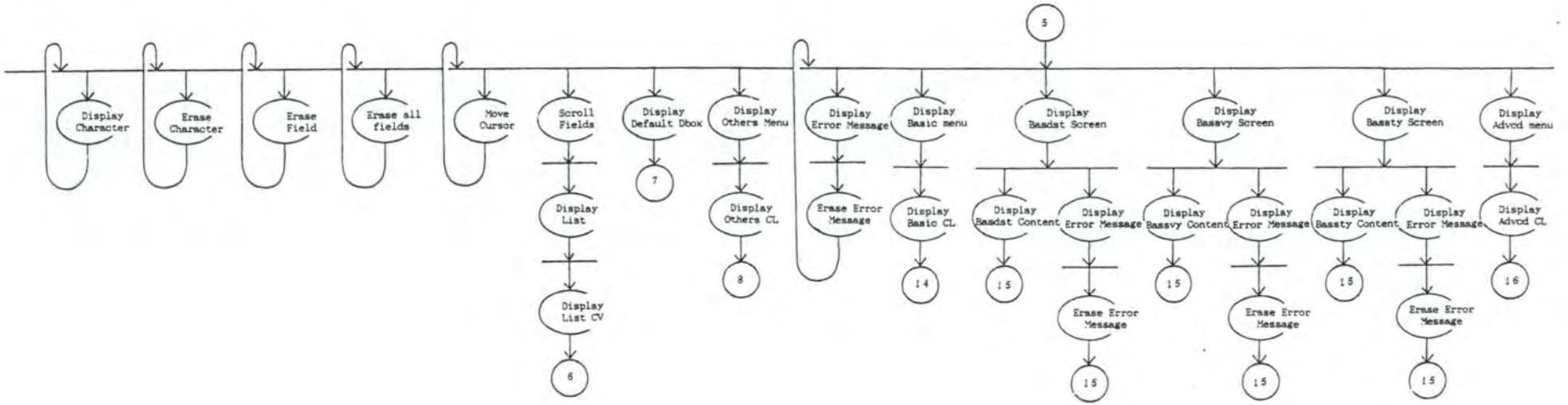


Figure 10.29 (I)

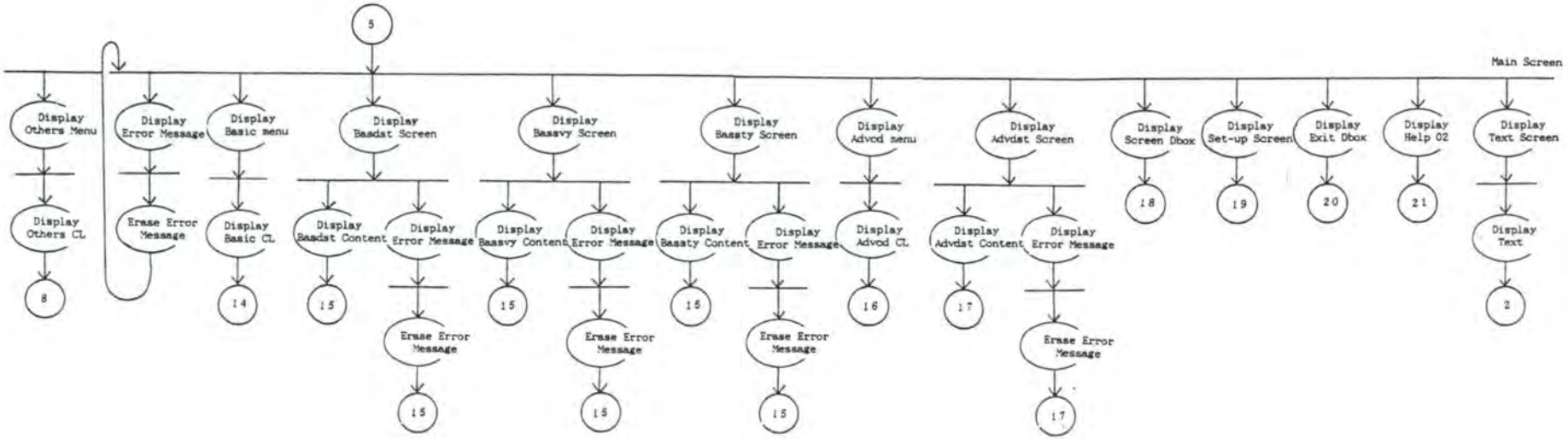


Figure 10.29 (II)

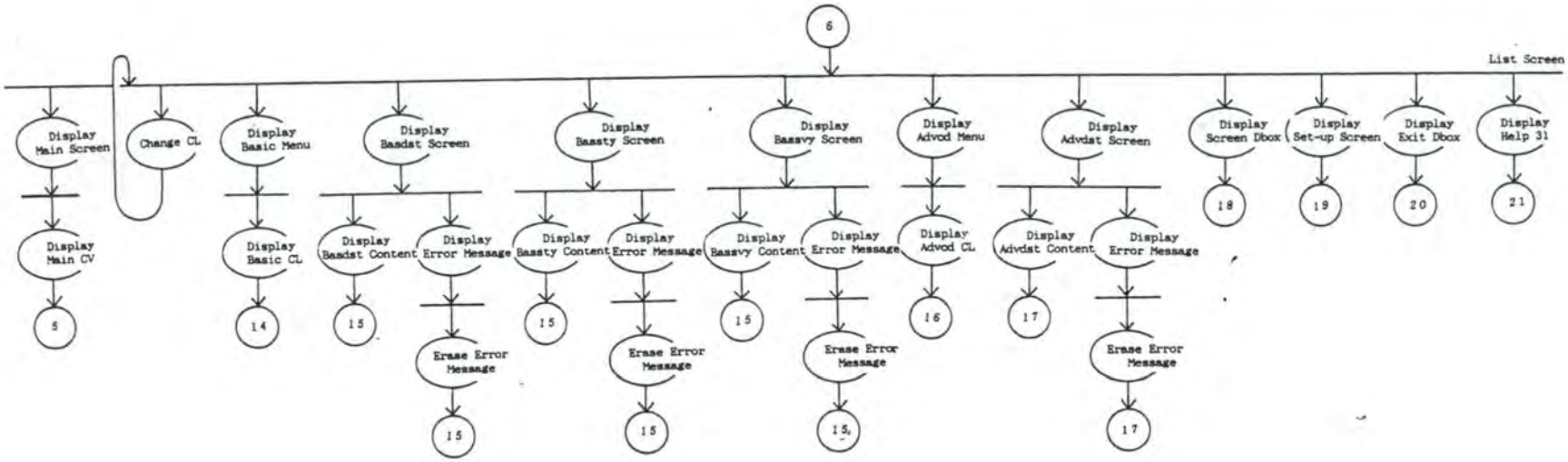

6
List Screen
Display Main Screen
Change CL
Display Basic Menu
Display Basdat Screen
Display Bassty Screen
Display Bassvy Screen
Display Advod Menu
Display Advdst Screen
Display Screen Dbox
Display Set-up Screen
Display Exit Dbox
Display Help 31
Display Main CV
Display Basic CL
Display Basdat Content
Display Error Message
Display Bassty Content
Display Error Message
Display Bassvy Content
Display Error Message
Display Advod CL
Display Advdst Content
Display Error Message
Erase Error Message
Erase Error Message
Erase Error Message
Erase Error Message
5
14
15
15
15
15
15
16
17
17
18
19
20
21

Figure 10.30

La figure 10.29 présente le menu principal.

La figure 10.30 représente l'option List.

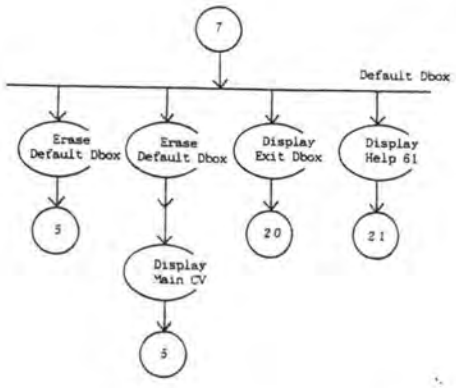


Figure 10.31

La figure 10.31 présente l'option Default.

La figure 10.32 représente l'option Others.

La figure 10.33 présente l'option Path. Notons que les options Byyear, Record, Freq et Posi sont tout à fait semblables à celle-ci. Il convient cependant de remplacer le numéro 9 par respectivement les numéros 10, 11, 12 et 13.

La figure 10.34 représente l'option Basic. Notons que l'option Advanced est tout à fait similaire à l'option Basic. Il est nécessaire de remplacer les numéros 14 et 15 par 16 et 17.

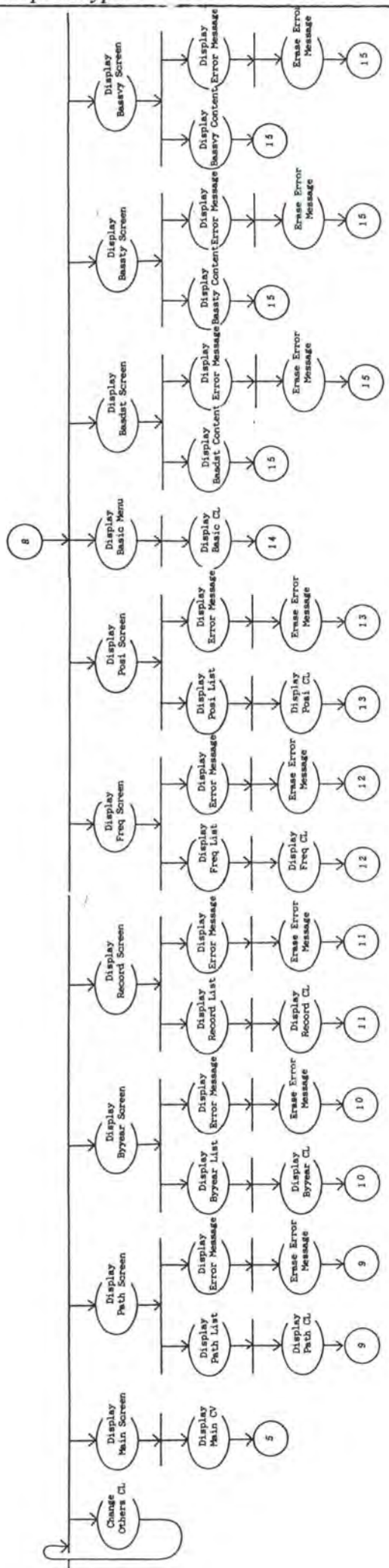


Figure 10.32 (I)

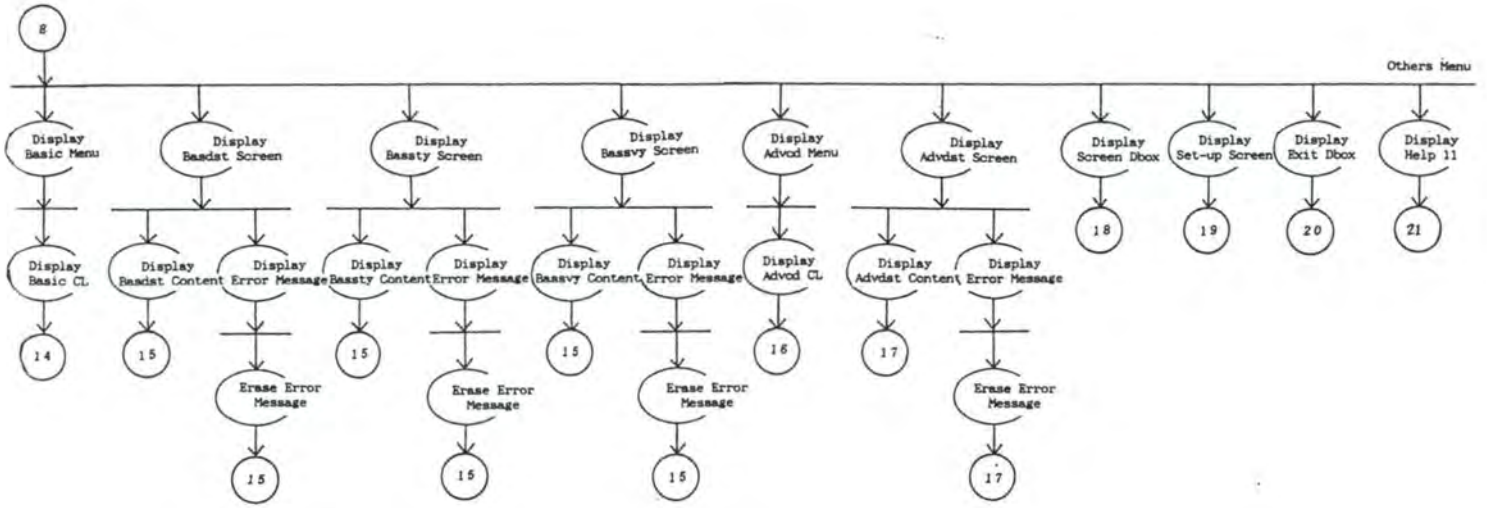


Figure 10.32 (II)

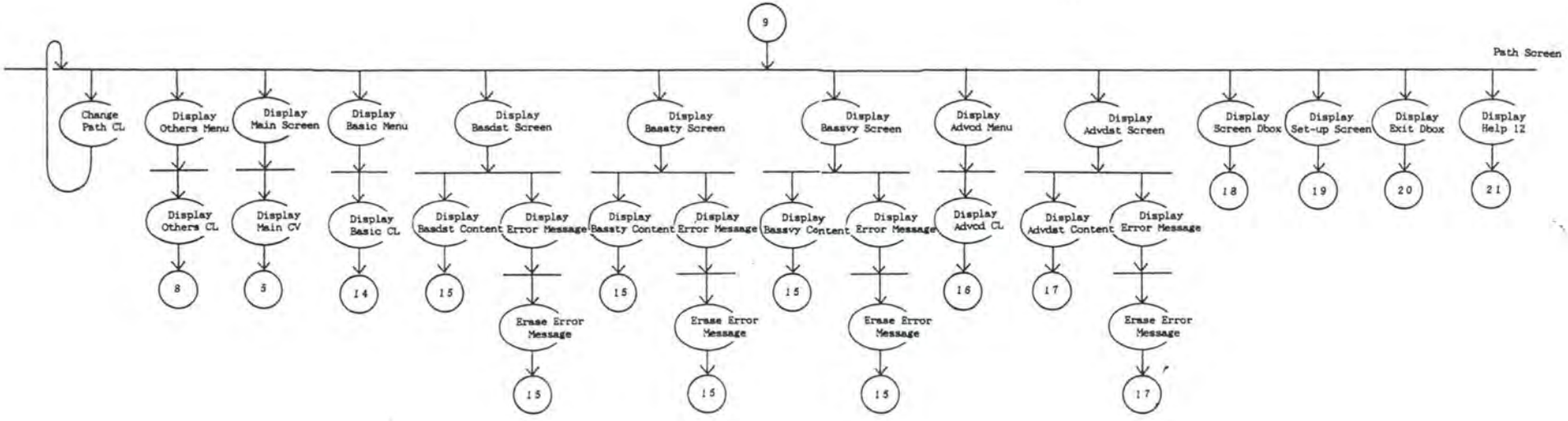

9
Path Screen
Change Path CL
Display Others Menu
Display Main Screen
Display Basic Menu
Display Basdst Screen
Display Basasty Screen
Display Basavy Screen
Display Advcd Menu
Display Advdst Screen
Display Screen Dbox
Display Set-up Screen
Display Exit Dbox
Display Help 12
Display Others CL
Display Main CV
Display Basic CL
Display Basdst Content
Display Error Message
Display Basasty Content
Display Error Message
Display Basavy Content
Display Error Message
Display Advcd CL
Display Advdst Content
Display Error Message
18
19
20
21
8
5
14
15
15
15
16
17
Erase Error Message
Erase Error Message
Erase Error Message
Erase Error Message
15
15
15
17

Figure 10.33

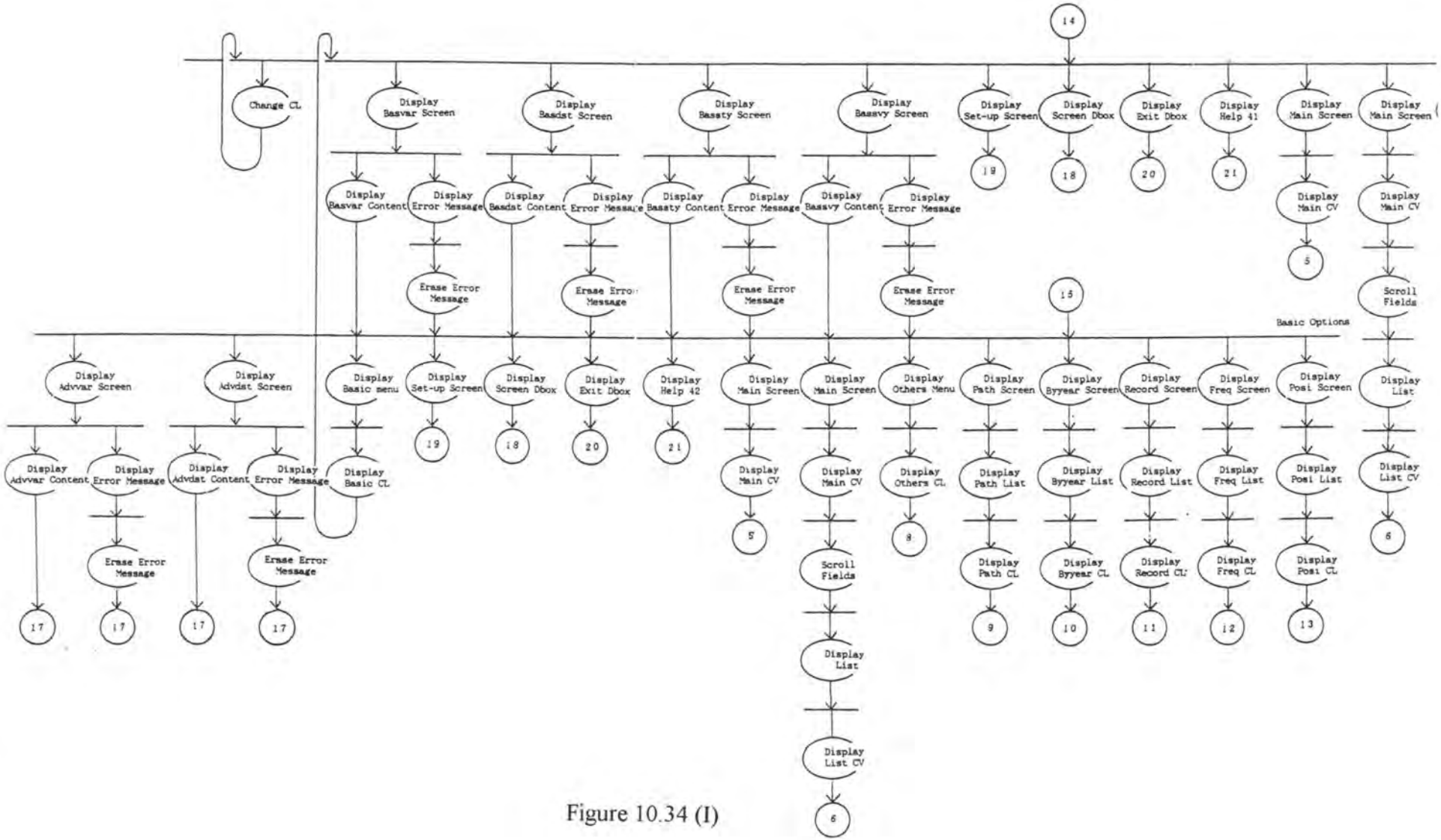


Figure 10.34 (I)

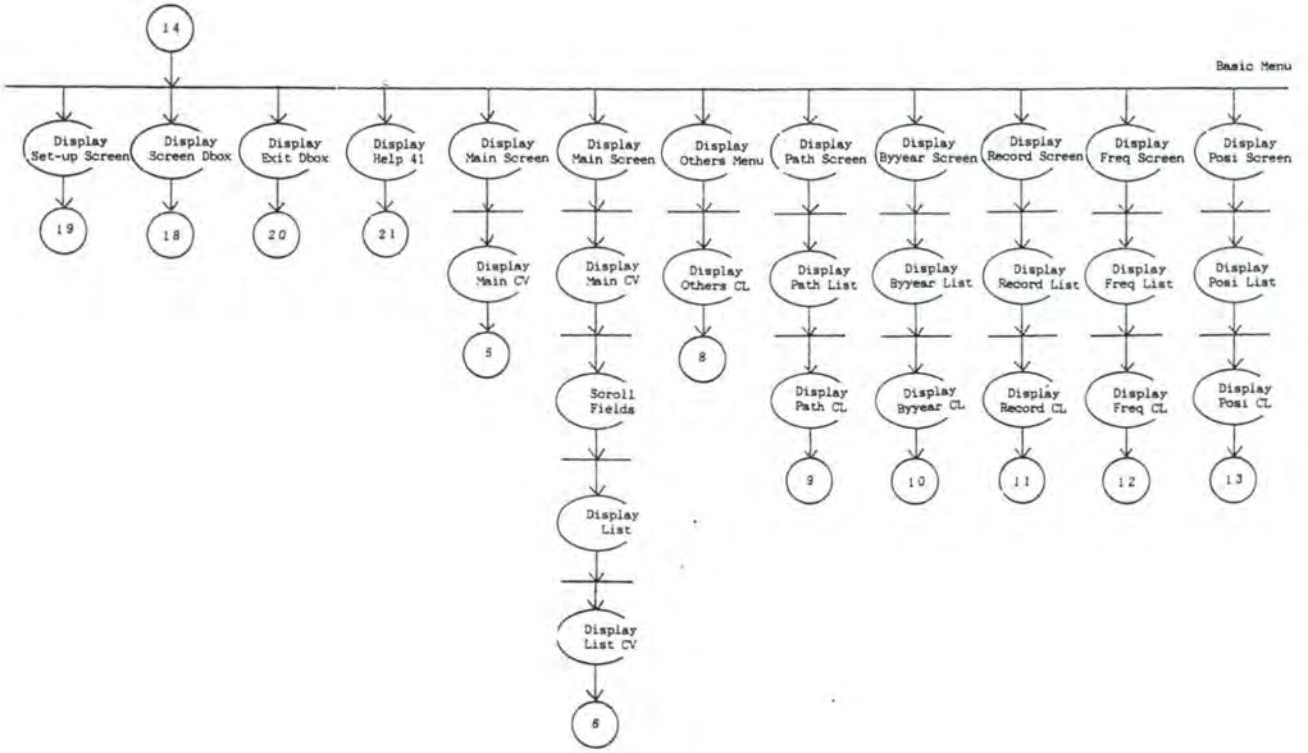


Figure 10.34 (II)

La figure 10.35 présente l'option Screen. Remarquons que cette option présente bien le choix de conception présenté au point 9.3. Notons également que nous n'avons pas repris tous les numéros de retour pour ne pas surcharger démesurément le schéma.

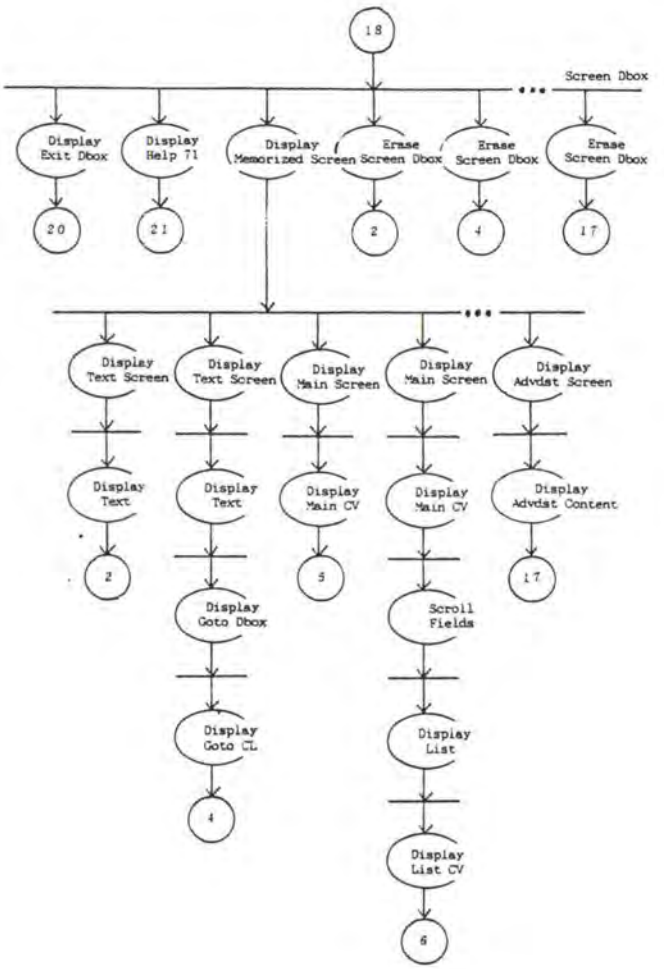


Figure 10.35

La figure 10.36 représente l'option Set-up. Notons que le retour de cette option est aussi abrégé.

La figure 10.37 présente l'option Exit.

La figure 10.38 représente l'option Help. Remarquons que le retour de cette option est également résumé.

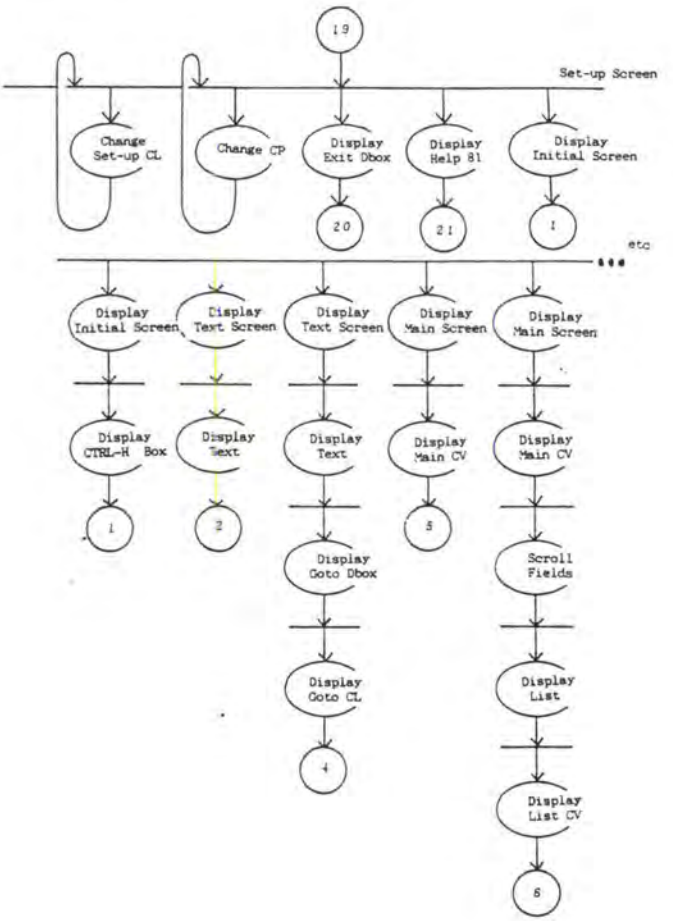


Figure 10.36

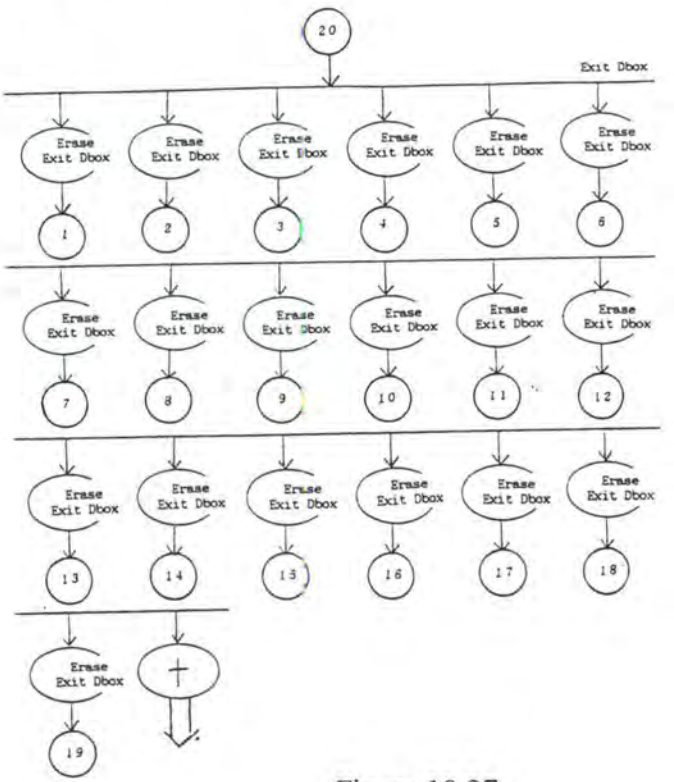


Figure 10.37

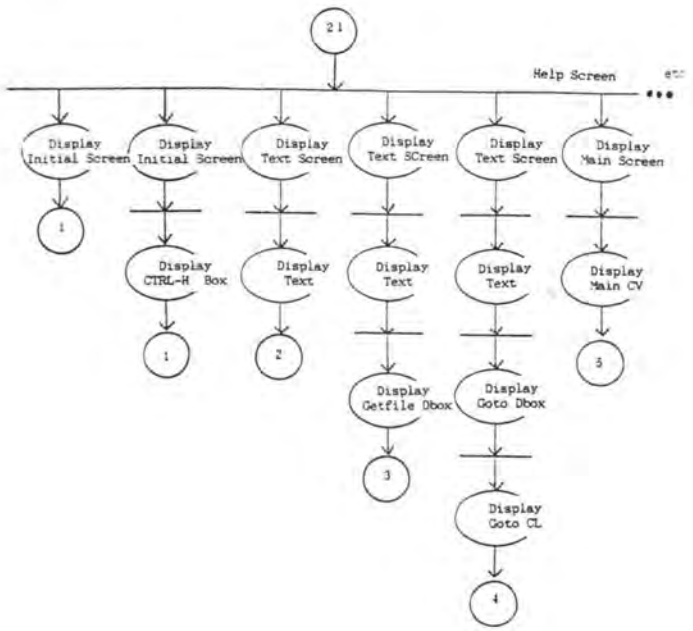


Figure 10.38

10.2.3 Comportement de l'objet Context Area

La figure 10.39 présente l'enchaînement des événements dans la zone de contexte. Remarquons que chaque situation (exceptée la première) correspond à un type de contexte différent (Texte, Main, GetFile, Double ou Help).

10.2.4 Comportement de l'objet Menu Bar

La figure 10.40 représente l'enchaînement des événements dans la barre de menu.

10.2.5 Comportement de l'objet Current Values

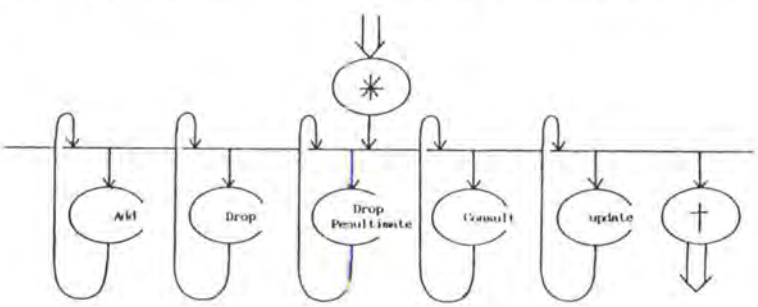


Figure 10.41

La figure 10.41 présente l'enchaînement des événements de la valeur courante.

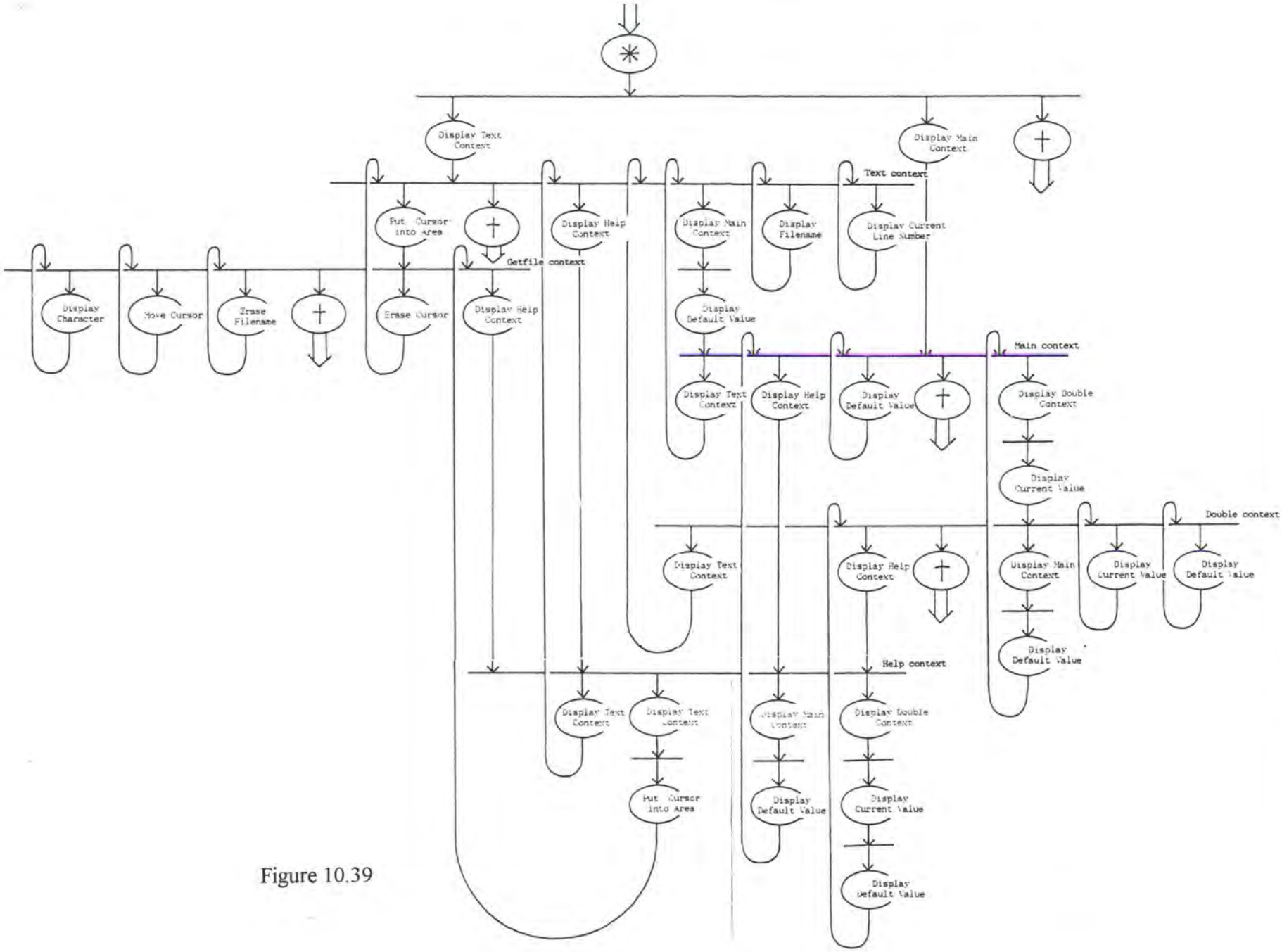

Display Text Context
Display Main Context
Text context
Put Cursor into Area
Display Help Context
Display Main Context
Display Filename
Display Current Line Number
Getfile context
Display Character
Move Cursor
Erase Filename
Erase Cursor
Display Help Context
Display Default Value
Main context
Display Text Context
Display Help Context
Display Default Value
Display Double Context
Display Current Value
Double context
Display Text Context
Display Help Context
Display Main Context
Display Current Value
Display Default Value
Display Default Value
Help context
Display Text Context
Display Text Context
Display Main Context
Display Double Context
Put Cursor into Area
Display Default Value
Display Current Value
Display Default Value

Figure 10.39

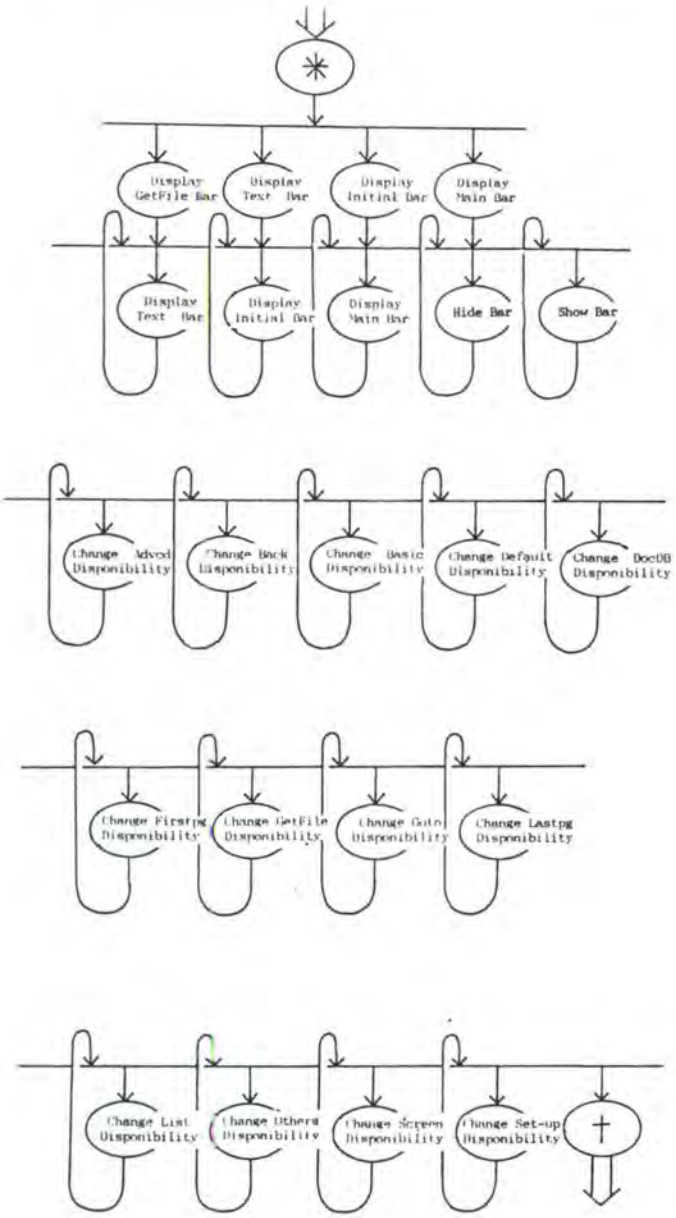


Figure 10.40

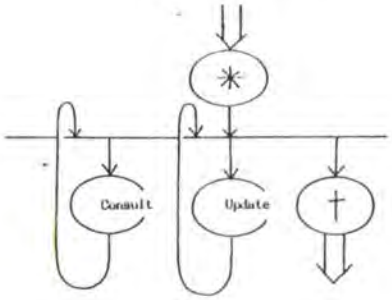


Figure 10.42

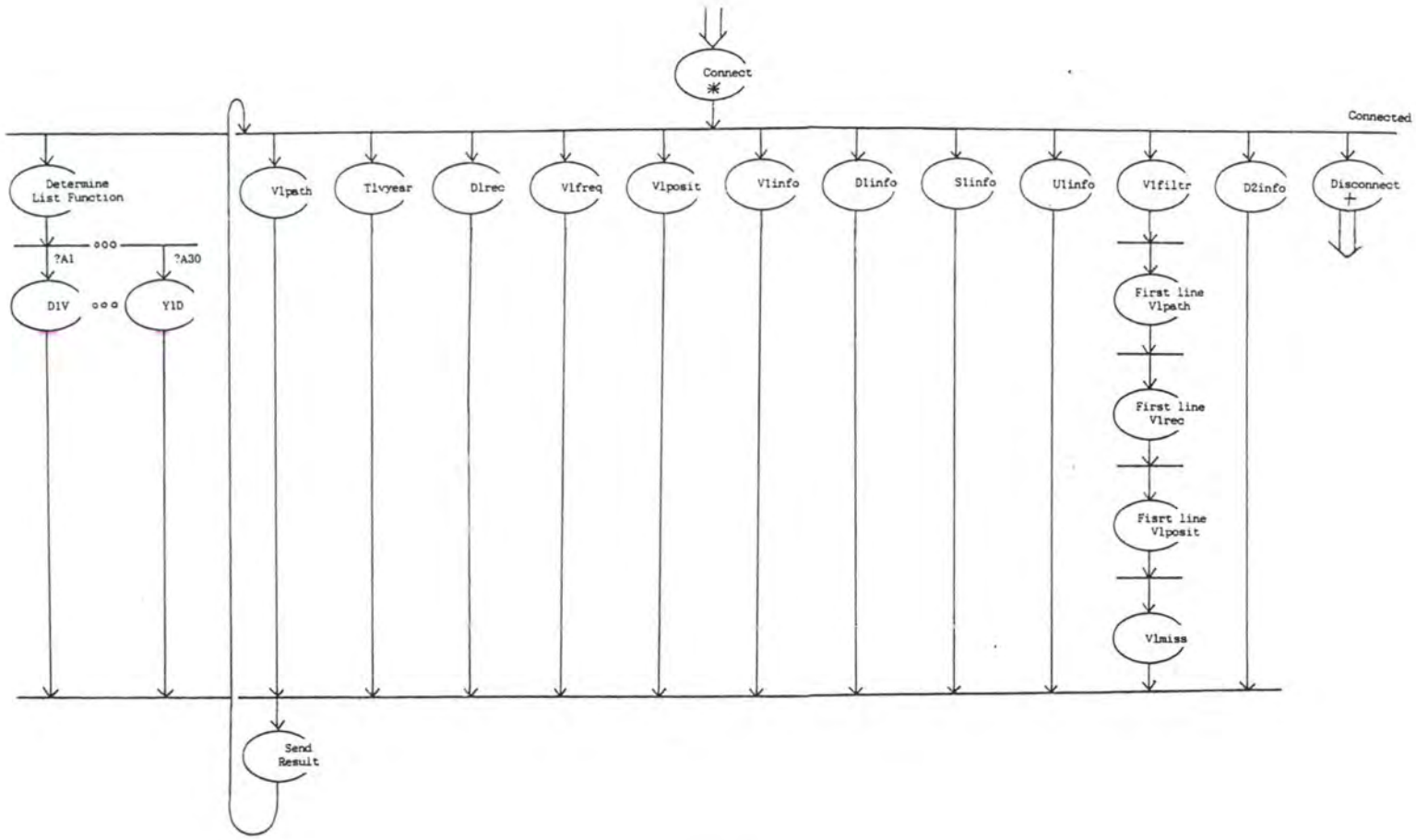

Connect
*
Connected
Determine
List Function
?A1
?A30
D1V
Y1D
V1path
T1vyear
D1rec
V1freq
V1posit
V1info
D1info
S1info
U1info
V1filtr
D2info
Disconnect
+
First line
V1path
First line
V1rec
Fisrt line
V1posit
V1miss
Send
Result

Figure 10.43

10.2.6 Comportement de l'objet Default Value

La figure 10.42 représente l'enchaînement des événements de la valeur par défaut.

10.2.7 Comportement de l'objet Database

La figure 10.43 présente l'enchaînement des événements de la base de données. Notons que comme dans le diagramme matriciel, nous n'avons pas développé ici les fonctions de l'option List. Seule la première et la dernière y figurent. De la même manière, seule la première et la dernière condition sont reprises :

A1 : (num = 1)
A30 : (num = 30).

10.2.8 Comportement de l'objet File

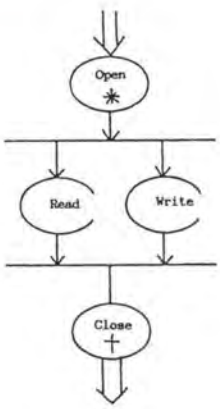


Figure 10.44

La figure 10.44 représente l'enchaînement des événements du fichier texte. La condition est :

A : open_error = True

10.2.9 Comportement de l'objet Text

La figure 10.45 présente l'enchaînement des événements du texte.

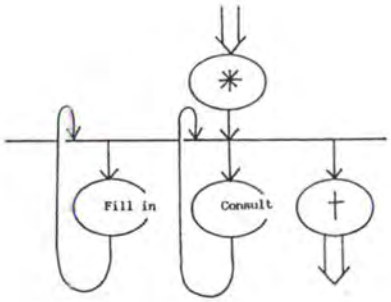


Figure 10.45

10.2.10 Comportement de l'objet List

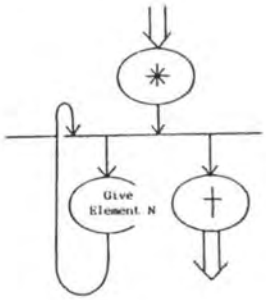


Figure 10.46

La figure 10.46 représente l'enchaînement des événements de la liste.

10.2.11 Comportement de l'objet Line

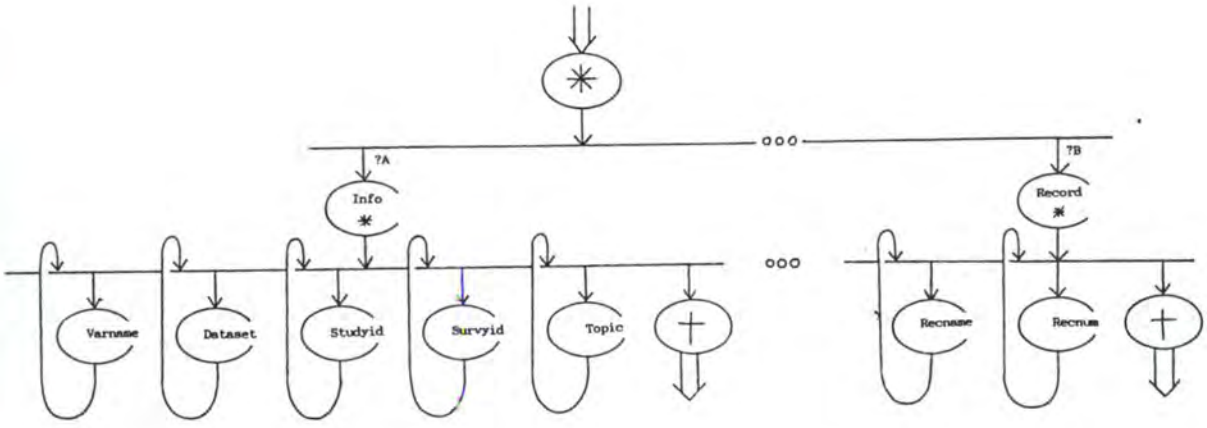


Figure 10.47

La figure 10.47 présente l'enchaînement des événements de la ligne. Notons que nous n'avons pas représenté le comportement de tous les objets Line composants mais seulement deux à titre d'exemples. Les conditions sont :

A : type = "info"
B : type = "rec"

10.3 Diagrammes d'initialisation

10.3.1 Initialisation de l'objet User

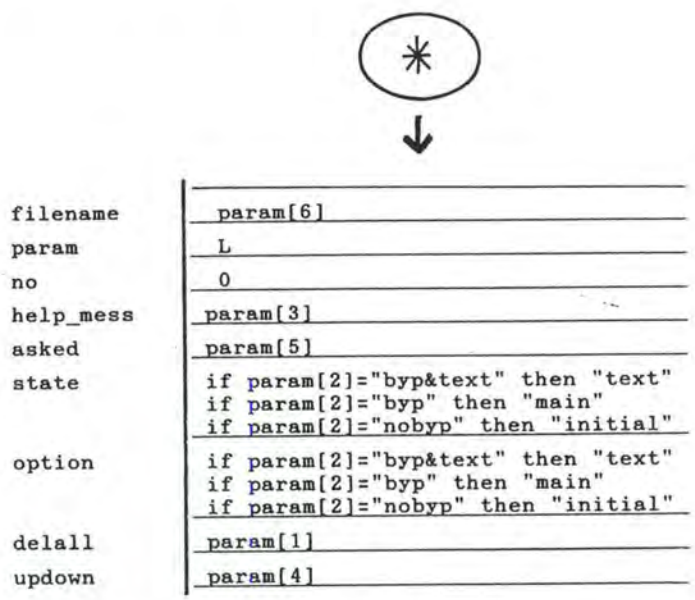


Figure 10.48

La figure 10.48 présente l'initialisation de l'utilisateur.

10.3.2 Initialisation de l'objet Working Area

La figure 10.49 représente l'initialisation de la zone de travail.

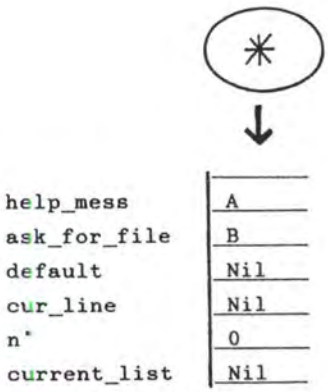


Figure 10.49

10.3.3 Initialisation de l'objet Context Area

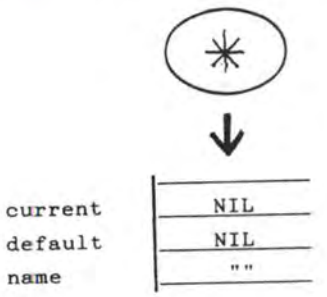


Figure 10.50

La figure 10.50 présente l'initialisation de la zone de contexte.

10.3.4 Initialisation de l'objet Menu Bar

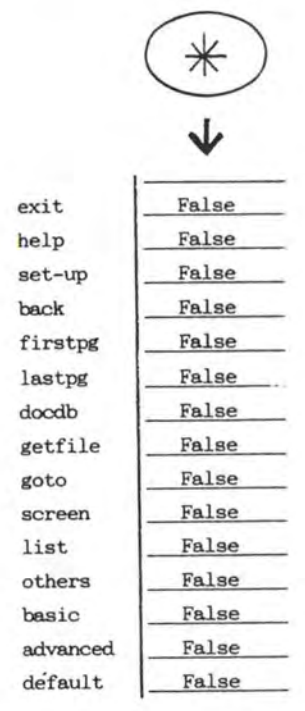


Figure 10.51

La figure 10.51 décrit l'initialisation de la barre de menu.

10.3.5 Initialisation de l'objet Current Values

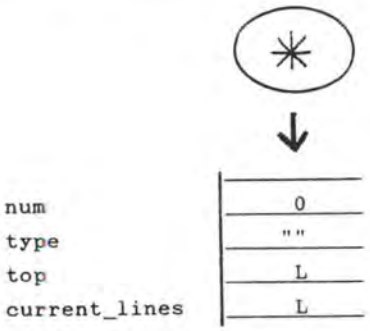


Figure 10.52

La figure 10.52 représente l'initialisation de la valeur courante.

10.3.6 Initialisation de l'objet Default Value

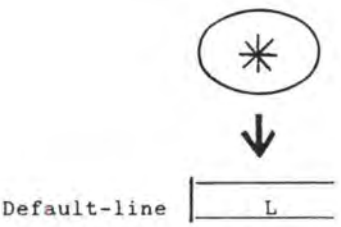


Figure 10.53

La figure 10.53 présente l'initialisation de la valeur par défaut.

10.3.7 Initialisation de l'objet Database

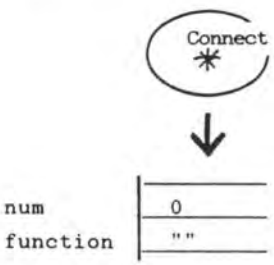


Figure 10.54

La figure 10.54 décrit l'initialisation de la base de données.

10.3.8 Initialisation de l'objet File

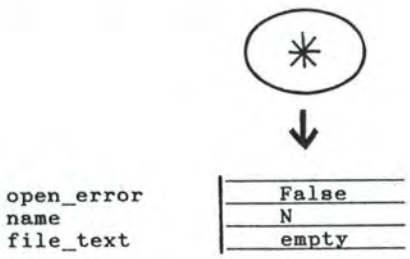


Figure 10.55

La figure 10.55 représente l'initialisation du fichier texte.

10.3.9 Initialisation de l'objet Text

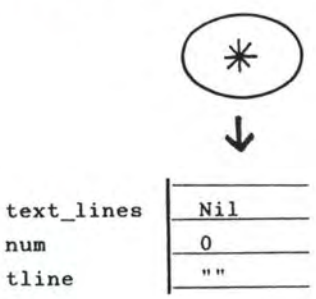


Figure 10.56

La figure 10.56 présente l'initialisation du texte.

10.3.10 Initialisation de l'objet List

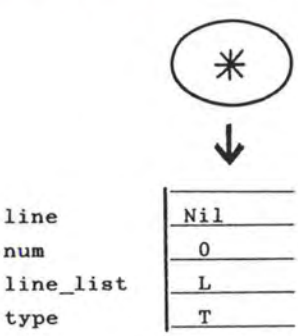


Figure 10.57

La figure 10.57 décrit l'initialisation de la liste.

10.3.11 Initialisation des objets Line

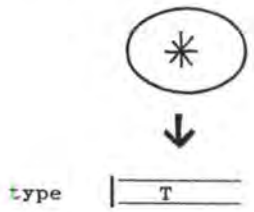


Figure 10.58

La figure 10.58 représente l'initialisation de la ligne.

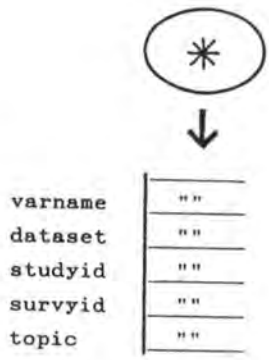


Figure 10.59

La figure 10.59 présente l'initialisation de l'objet Info Line. L'initialisation des autres types de lignes ne sont pas représentés ici mais sont similaires à Info Line.

10.4 Diagrammes d'interactions et de liaisons de valeurs

Pour présenter les interactions entre les événements des divers objets, nous avons décidé de choisir le comportement de l'objet User comme fil conducteur. Nous montrons d'abord sa naissance et les conséquences qu'elle a sur les autres objets. Nous présentons ensuite les interactions des objets pour les différentes options (y compris la sortie).

10.4.1 La naissance de l'application

La naissance de l'objet User est l'événement déclencheur de la naissance de l'application et donc de la plupart des objets (figure 10.60 à 10.63).

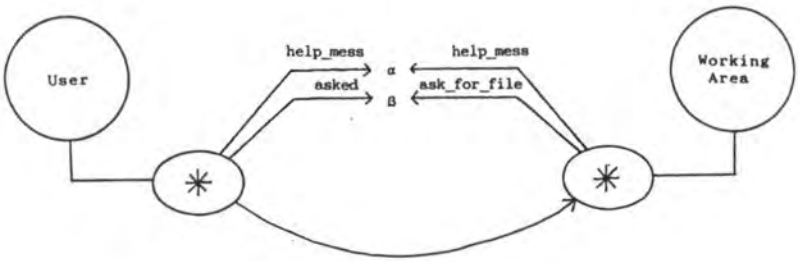


Figure 10.60

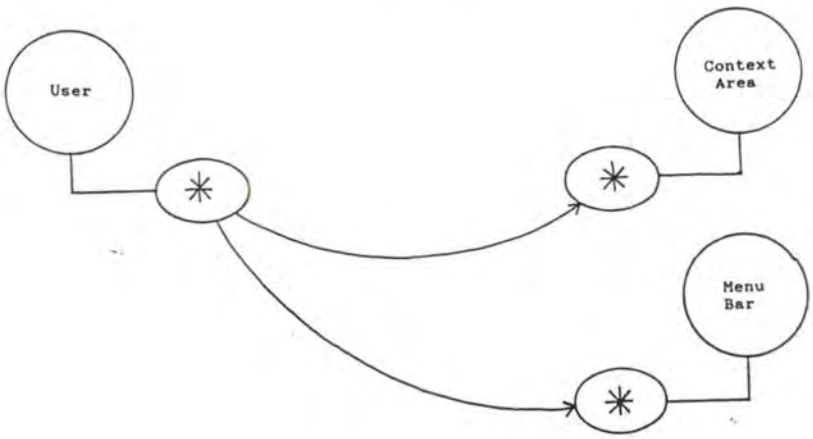


Figure 10.61

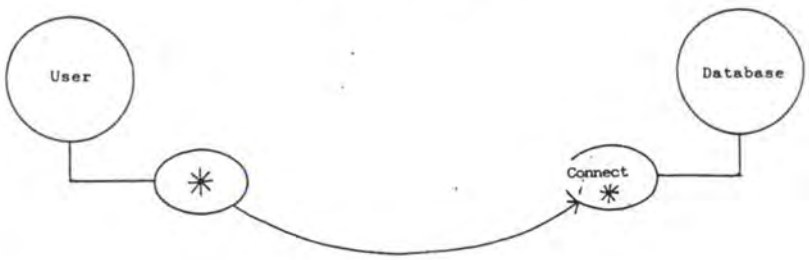


Figure 10.62

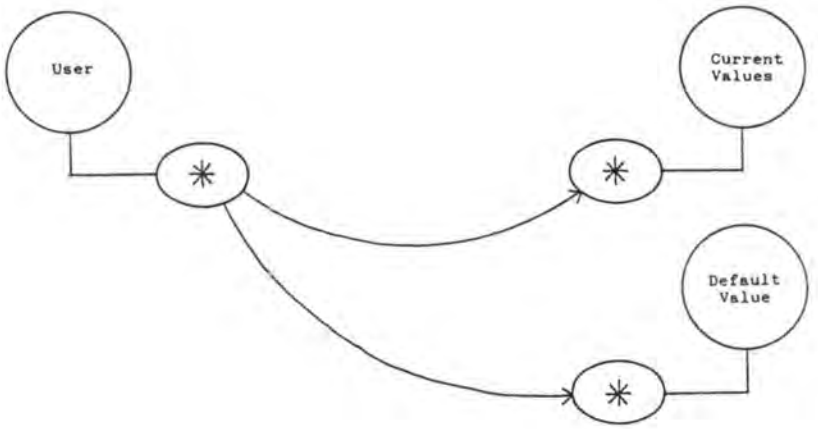


Figure 10.63

Les figures 10.64 et 10.65 présentent la lecture des paramètres de set-up.

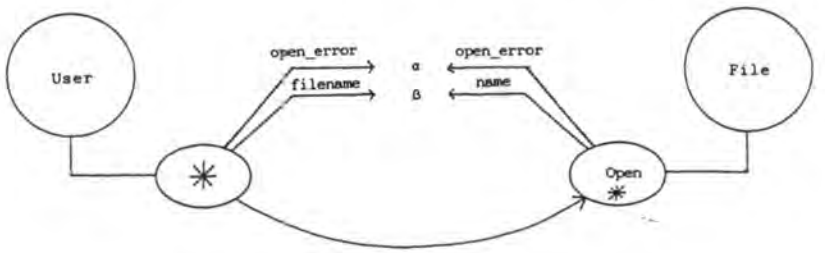


Figure 10.64

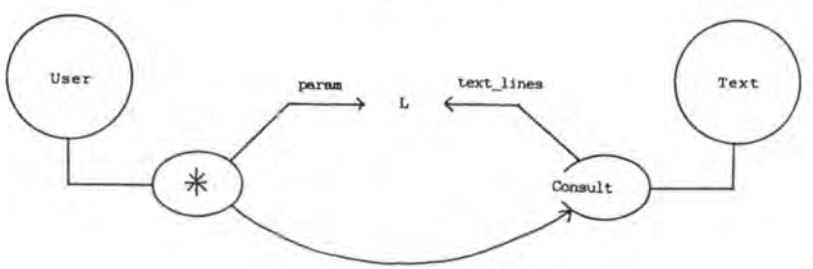


Figure 10.65

Les figures 10.66, 10.67, 10.68 et 10.73 représentent les différents débuts de session possibles, respectivement l'arrivée dans le menu initial, dans l'écran texte avec demande de fichier, dans l'écran texte sans demande de fichier et dans l'écran principal.

Les figures 10.69 à 10.72 expliquent comment le texte est chargé et affiché dans l'option texte. Pour le détail de fonctionnement du GetFile, veuillez vous reporter au point suivant.

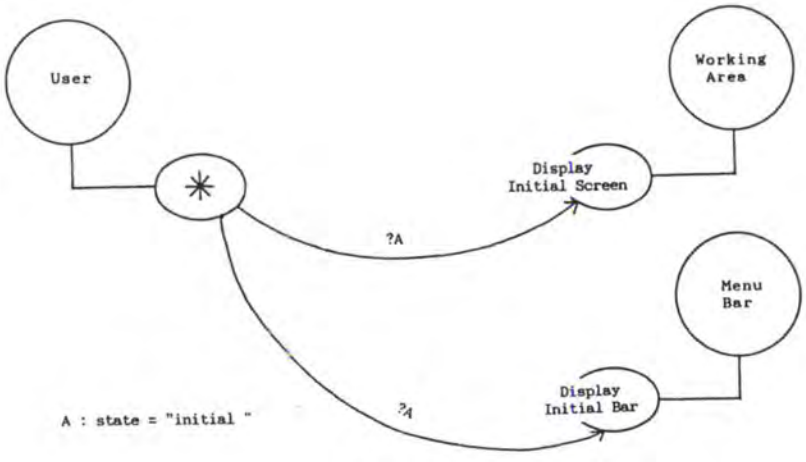


Figure 10.66

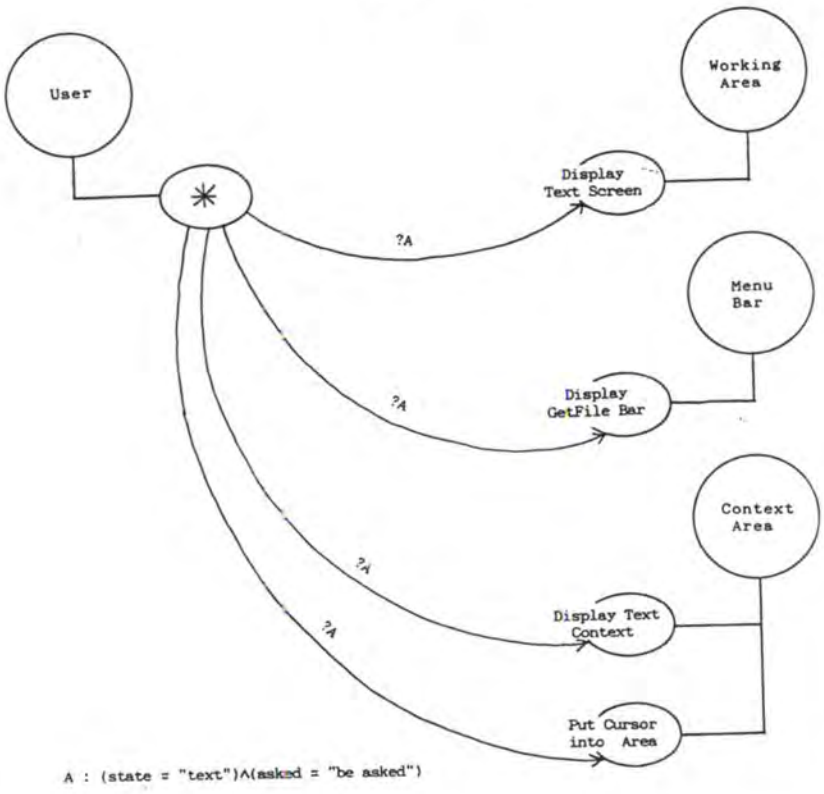


Figure 10.67

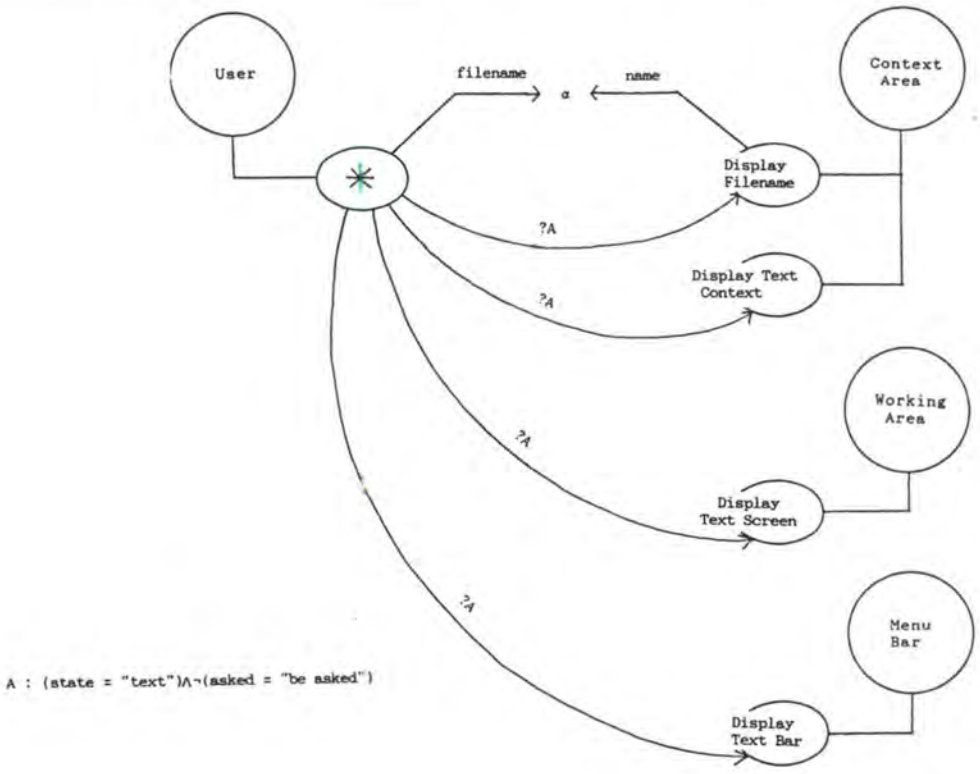


Figure 10.68

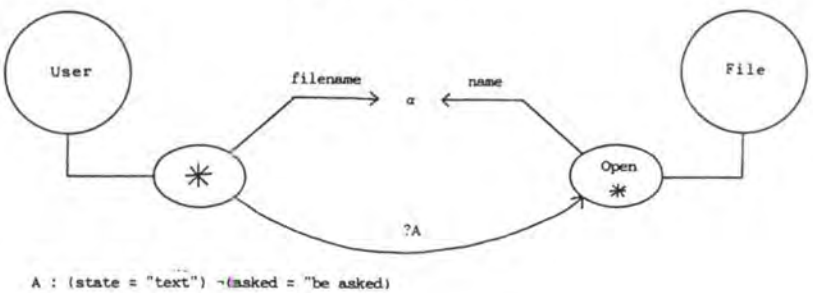


Figure 10.69

Figure 10.70

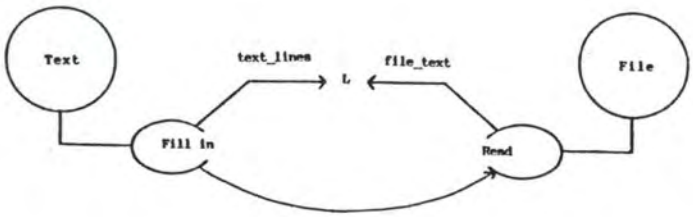


Figure 10.71

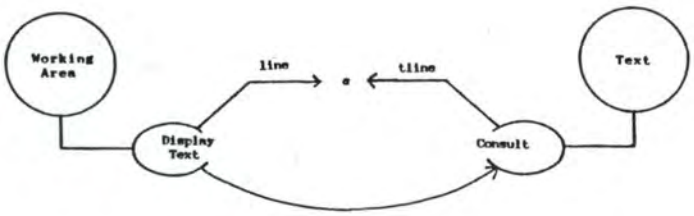


Figure 10.72

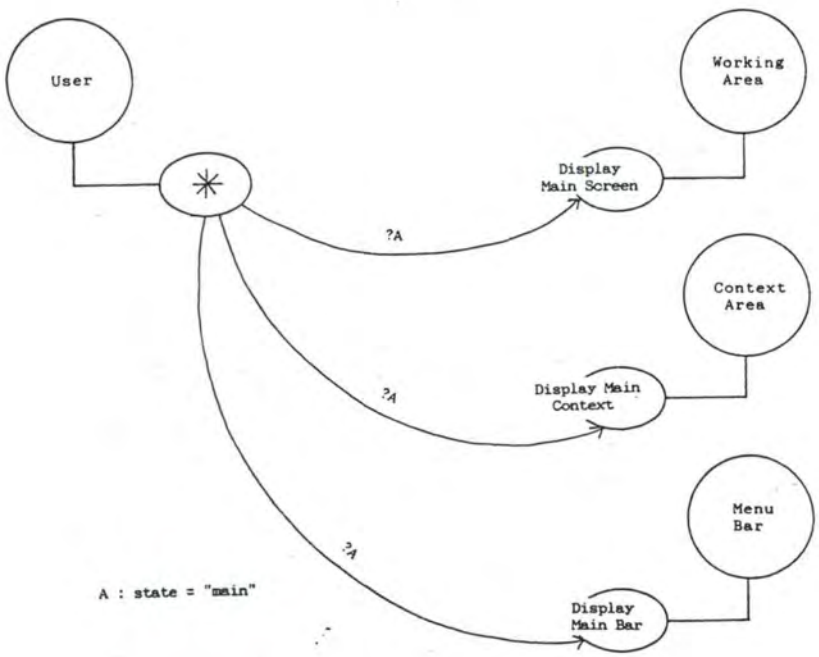


Figure 10.73

Les figures 10.74 et 10.75 décrivent le remplissage initial des valeurs courantes et par défaut.

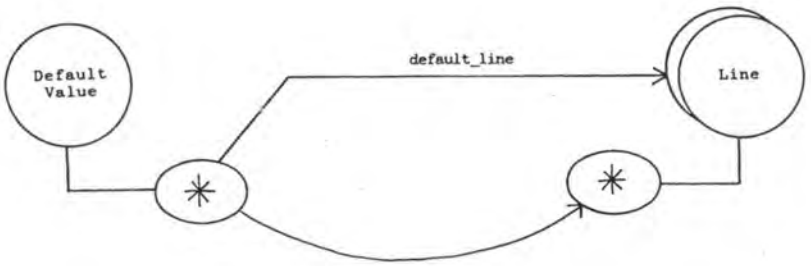


Figure 10.74

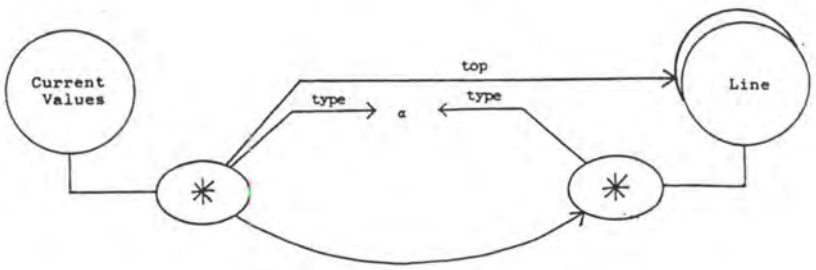


Figure 10.75

10.4.2 Le menu initial de l'application

La figure 10.76 présente le choix de l'option Only Docdb.

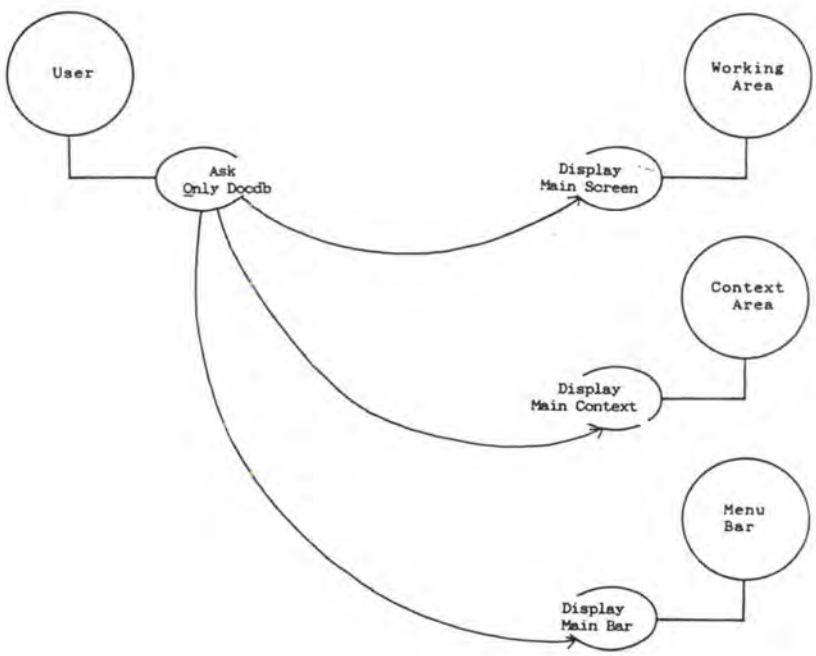


Figure 10.76

La figure 10.77 représente le choix de l'écran Text avec demande de fichier (voir ensuite le GetFile au point suivant).

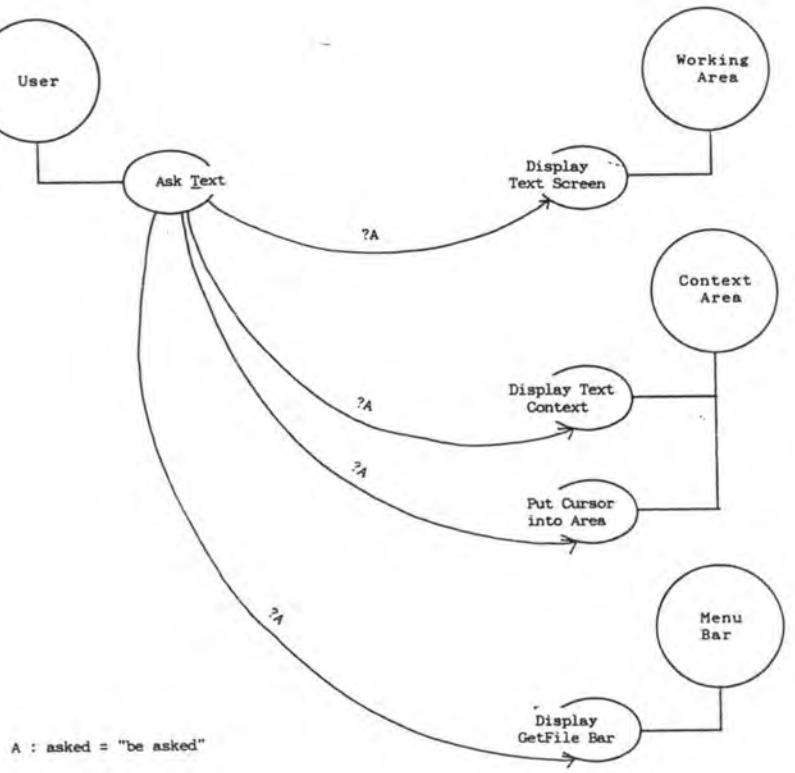


Figure 10.77

Les figures 10.78 et 10.79 décrivent le choix de l'écran Text avec récupération automatique du dernier fichier utilisé.

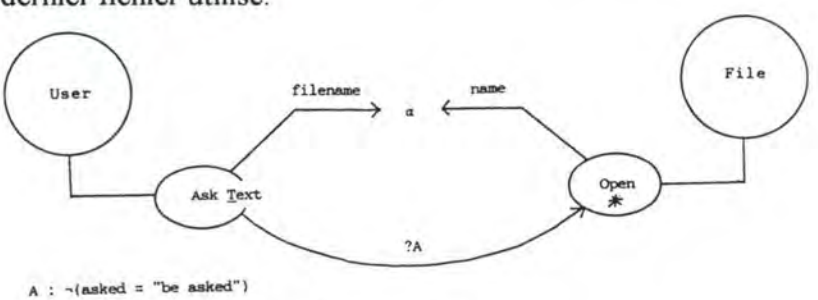


Figure 10.78

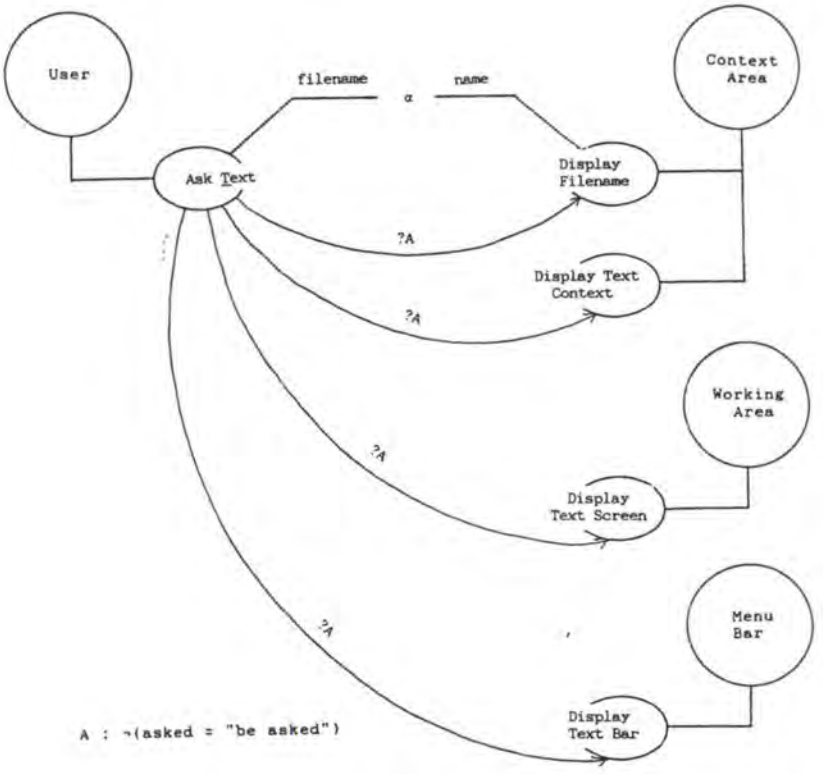


Figure 10.79

Le menu initial autorise également un appel aux options Set-up, Help et Exit. Nous les envisageons plus loin.

10.4.3 L'écran Text

La figure 10.80 présente le déplacement au sein du texte.

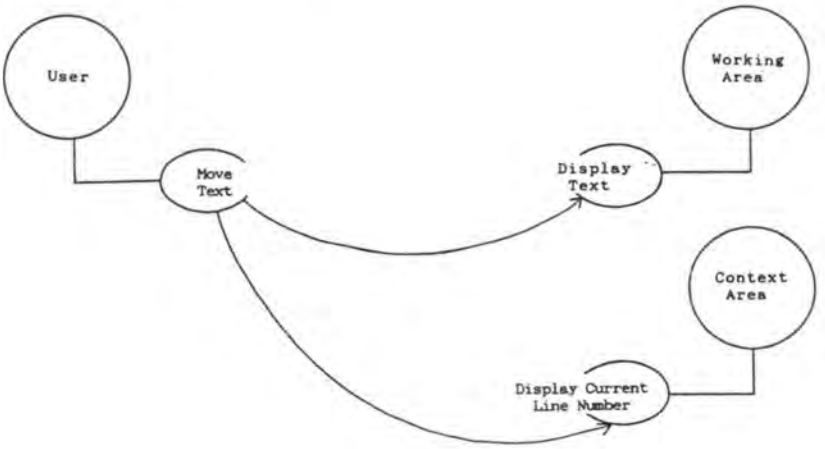


Figure 10.80

Les figures 10.81 à 10.84 décrivent les événements de l'option GetFile. Rappelons que les interactions avec les objets Text et File sont déjà reprises au point précédent.

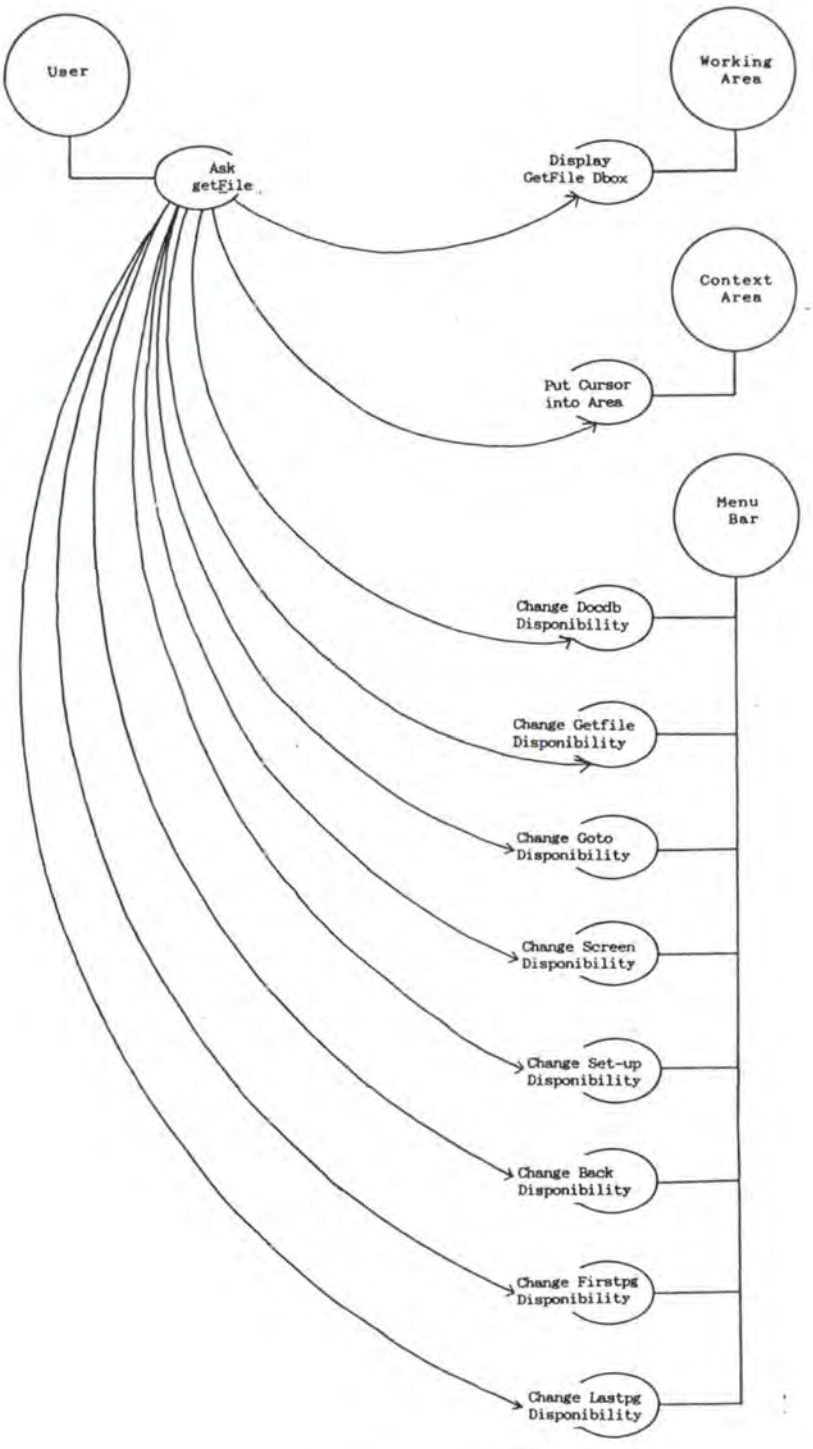


Figure 10.81

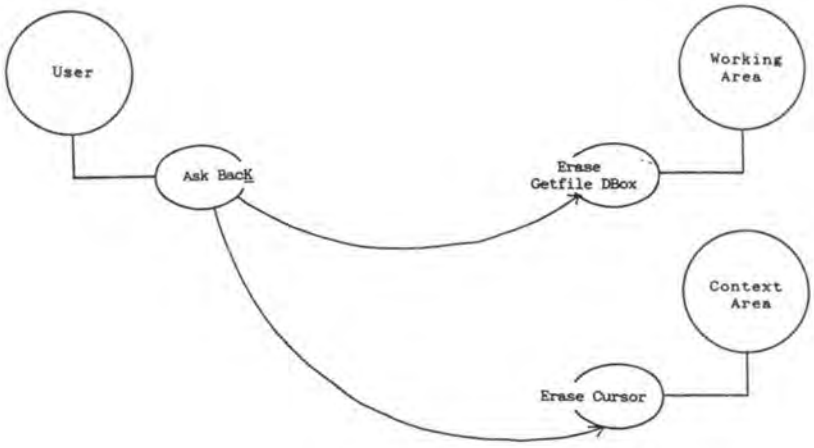


Figure 10.82

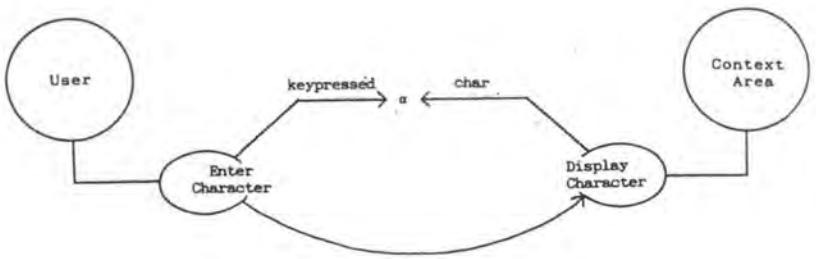


Figure 10.83

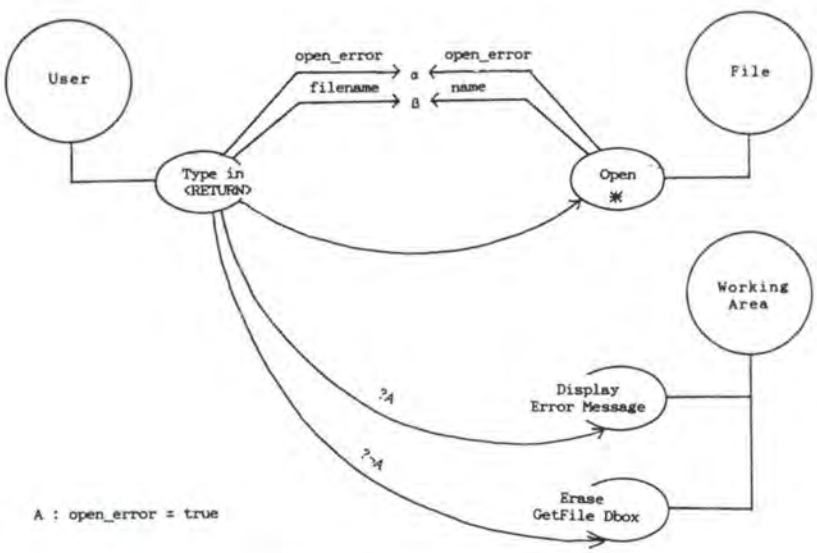


Figure 10.84

Remarquons que les interactions pour l'option Goto sont tout à fait similaires à celles de GetFile. Notamment, "Ask Lastpg" et "Ask Firstpg" ont un effet semblable à "Move Text" dans la zone de travail.

Notons que nous ne détaillerons plus tous les changements de disponibilité de la barre de menu plus loin.

L'écran texte et ses options autorisent également des appels aux options Screen, Set-up, Help et Exit. Le lecteur est invité à se reporter aux options respectives.

10.4.4 L'écran Main

Les figures 10.85 à 10.87 décrivent le passage de l'écran texte à l'écran principal.

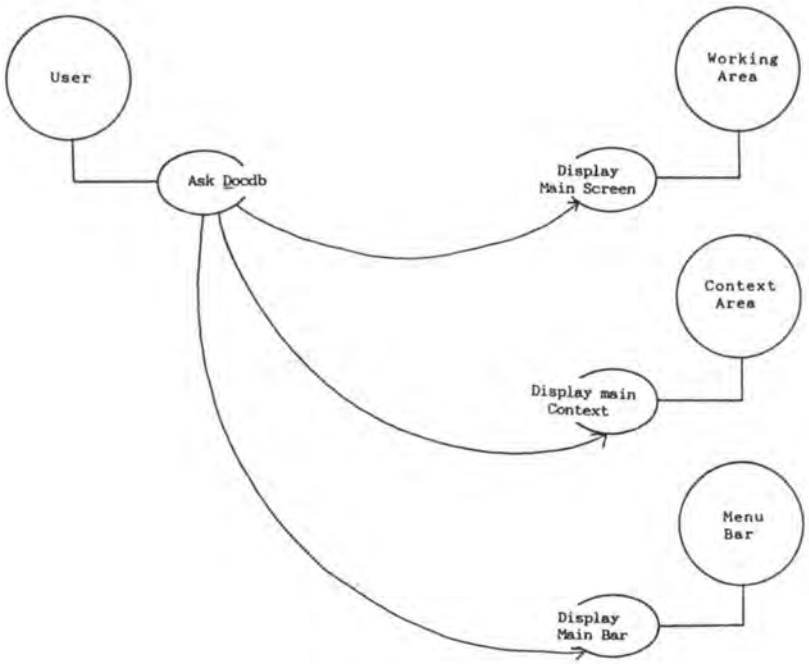


Figure 10.85

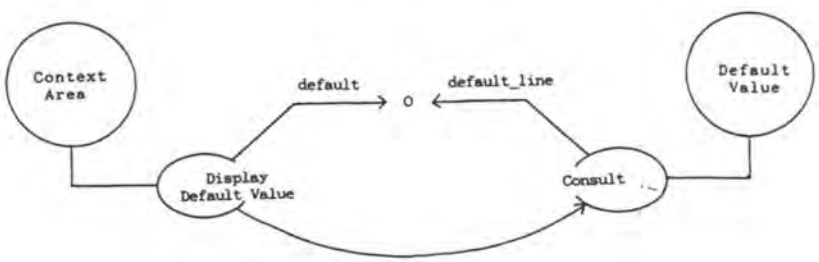


Figure 10.86

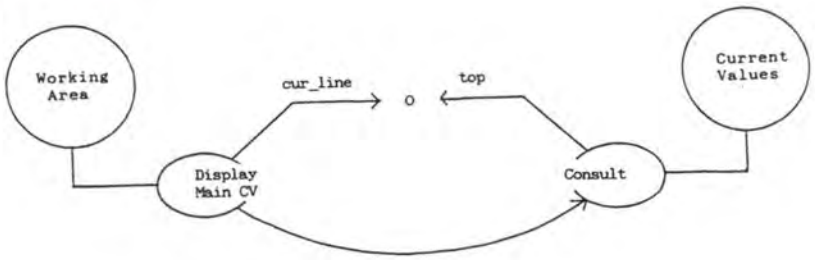


Figure 10.87

La figure 10.88 présente le déplacement du curseur dans les champs.

Figure 10.88

La figure 10.89 explique le passage de l'écran principal à l'écran texte via l'option "Ask eXit".

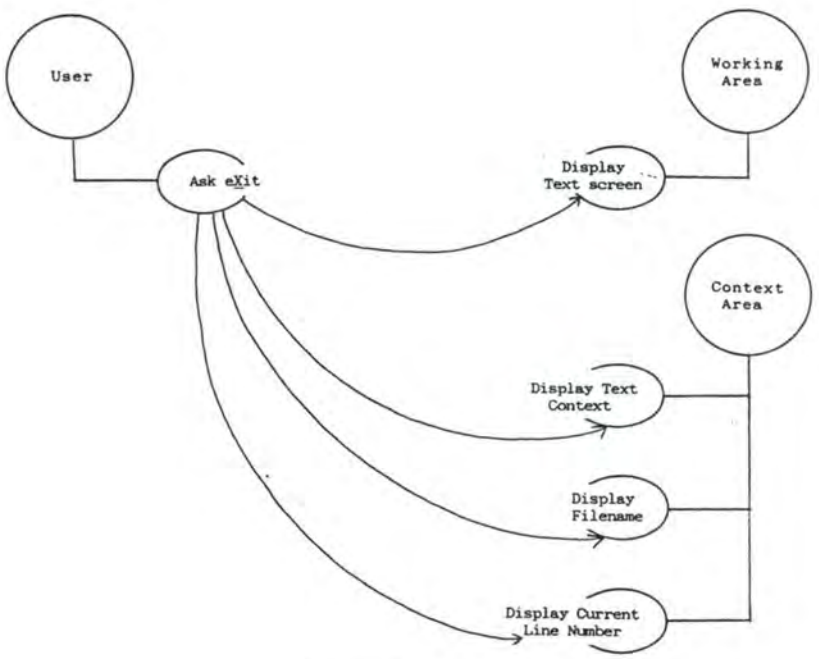


Figure 10.89

L'écran principal autorise également des appels aux options Advanced, Basic, List, Others, Default, Screen, Set-up, Help et Exit. Le lecteur est invité à se reporter aux options respectives. Rappelons que les changements de barre de menu ne sont plus détaillés.

10.4.5 Les options Basic et Advanced

Nous ne présentons ici que les interactions de Basic, mais celles d'Advanced sont similaires.

Les figures 10.90 et 10.91 représentent l'appel au menu Basic.

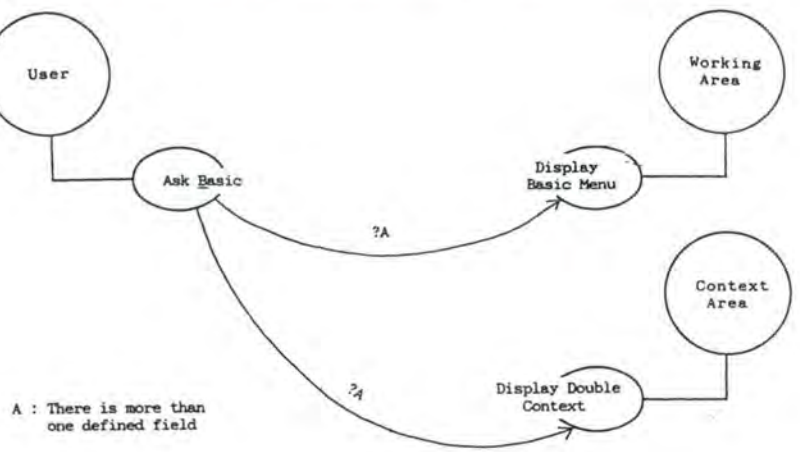


Figure 10.90

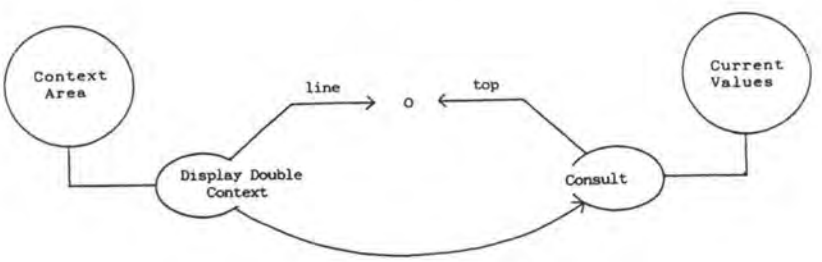


Figure 10.91

La figure 10.92 décrit un retour du menu Basic vers l'écran principal. Notons que si l'option Basic a été appelée d'un autre écran que le principal, il convient de réafficher l'écran approprié.

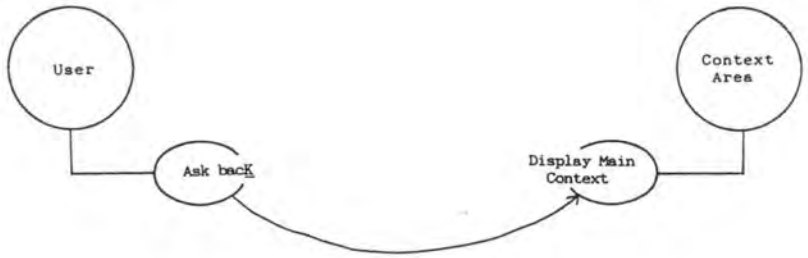


Figure 10.92

La figure 10.93 exprime le changement de ligne dans le menu.

Figure 10.93

La figure 10.94 présente le choix d'une option dans le menu.

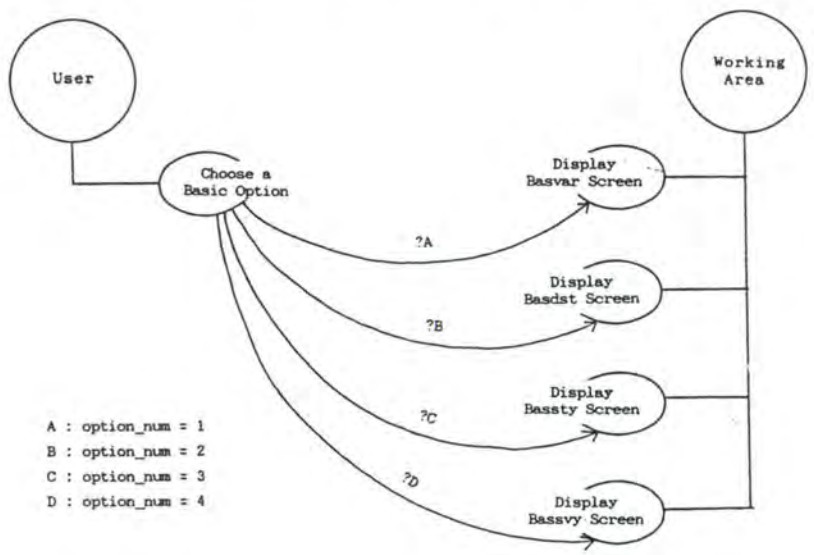


Figure 10.94

La figure 10.95 représente l'appel à l'option Advanced à partir d'un des écrans Basic. Rappelons qu'il n'est pas possible de demander ces informations pour l'étude et l'enquête. La figure 10.96 décrit le changement de disponibilité dans la barre de menu qui traduit ce dernier cas.

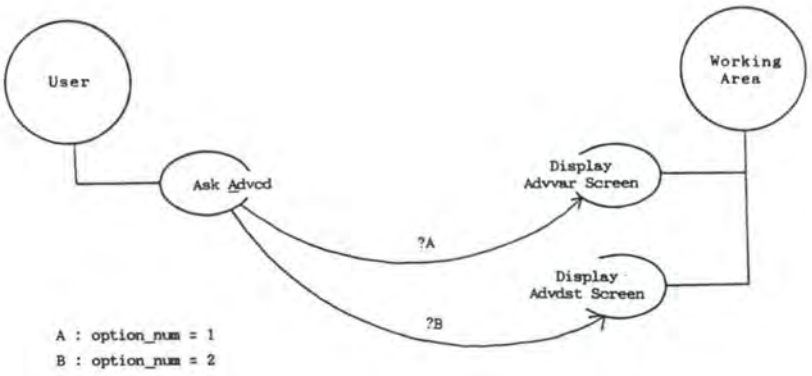


Figure 10.95

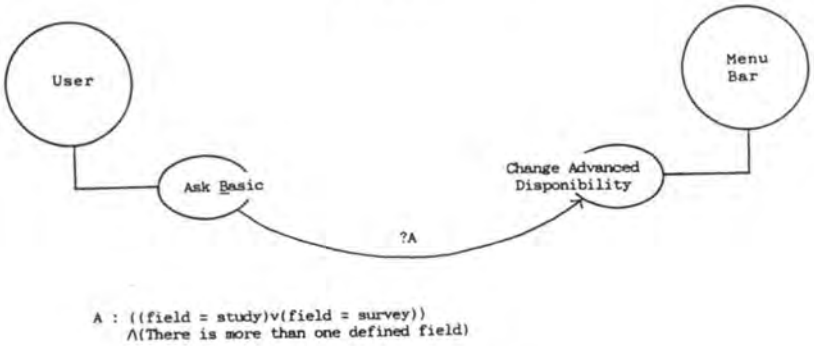


Figure 10.96

La figure 10.97 représente l'appel direct à une option Basic.

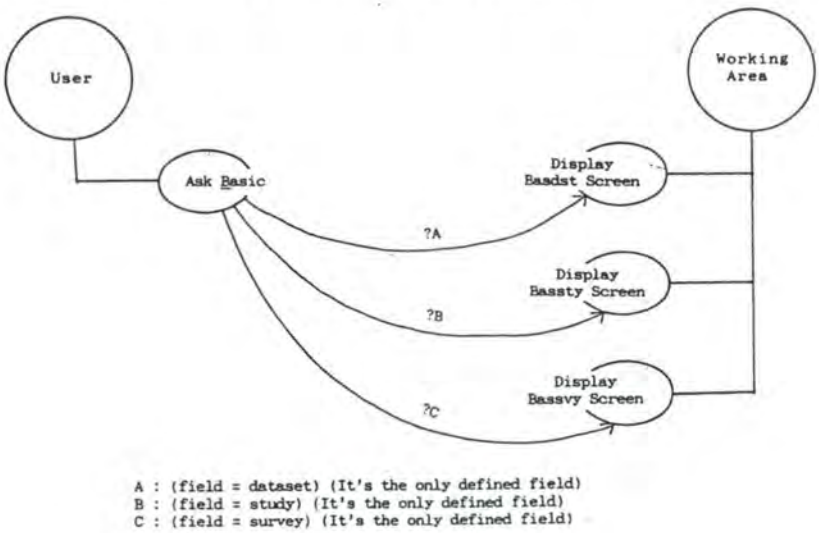


Figure 10.97

Lorsque l'on quitte une option Basic pour revenir au menu Basic, la liste courante de la zone de travail (qui contenait les informations affichées) est détruite. La figure 10.98 illustre ce cas.

Figure 10.98

Les écrans Basic et Advanced autorisent également des appels aux options Screen, Set-up, Help et Exit. Le lecteur est invité à se reporter aux options respectives. Rappelons que les changements de barre de menu ne sont plus détaillés.

10.4.6 L'option List

Les figures de 10.99 à 10.103 décrivent l'appel à l'option List et l'affichage de celle-ci. En particulier, la figure 10.99 présente le déroulement des boîtes et l'appel à la base de données, la figure 10.100 l'affichage des résultats produits par la base de données et les figures 10.101 à 10.103 le changement et l'affichage contrasté de la valeur courante.

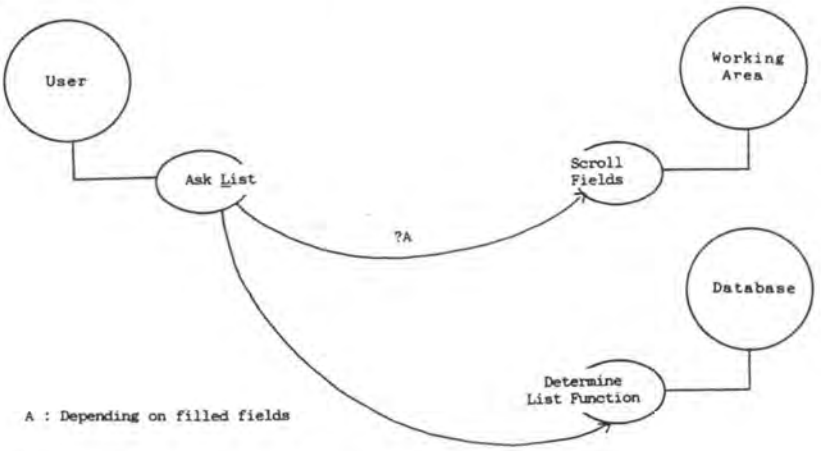


Figure 10.99

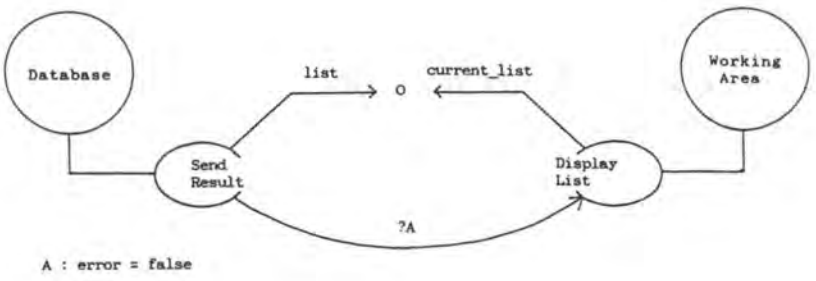


Figure 10.100

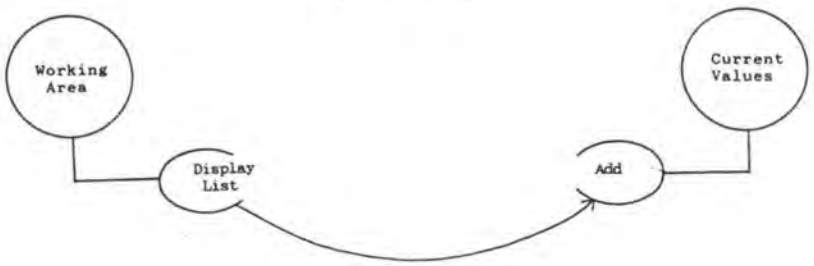


Figure 10.101

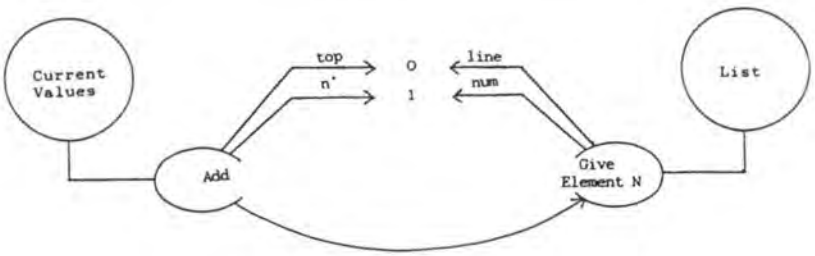


Figure 10.102

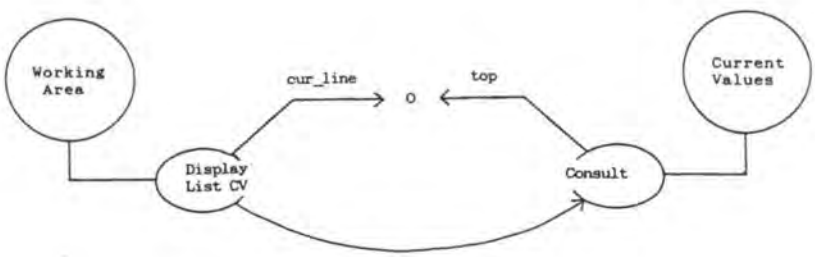


Figure 10.103

Les figures 10.104 à 10.106 montrent le déplacement dans la liste et le changement de valeur courante qui en résulte.

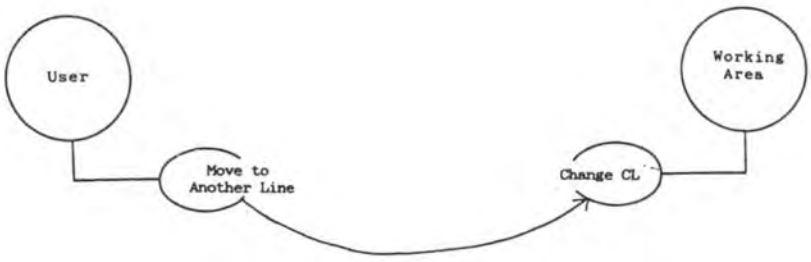


Figure 10.104

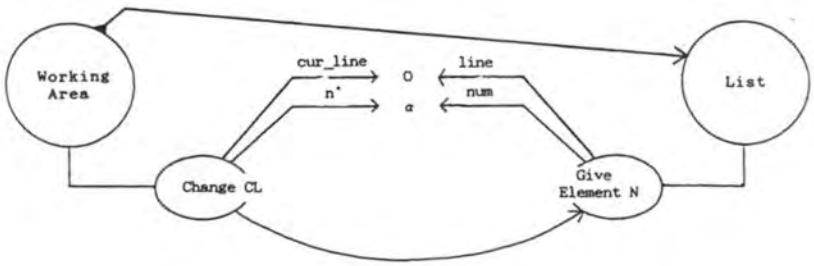


Figure 10.105

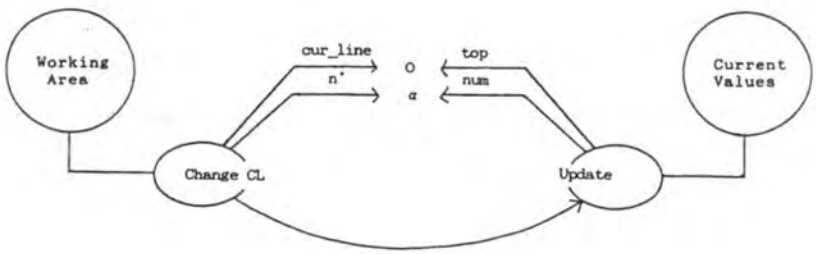


Figure 10.106

La figure 10.107 présente la sélection de l'élément courant.

Figure 10.107

L'écran List autorise également des appels aux options Basic, Advanced, Default, Screen, Set-up, Help et Exit. Le lecteur est invité à se reporter aux options respectives. Rappelons que les changements de barre de menu ne sont plus détaillés.

10.4.7 L'option Others

Nous ne présentons ici que les interactions de la fonction Path mais celles de Byyear, Freq, Record et Posi sont similaires. Notons que l'appel au menu Others et le comportement de celui-ci sont identiques à ceux des options Basic et Advanced. La figure 10.108 précise le choix d'une option Others.

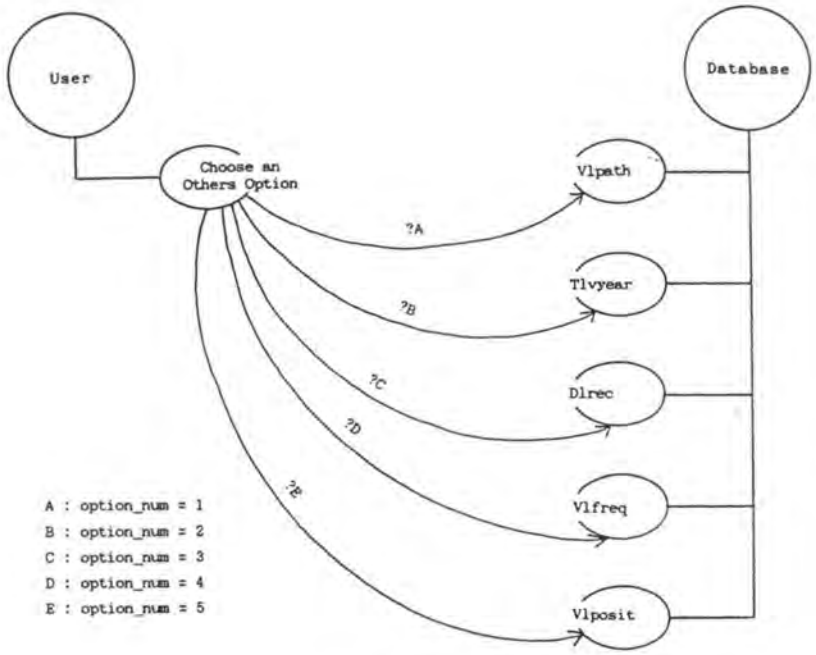


Figure 10.108

Les figures 10.109 à 10.111 représentent l'affichage de la liste et ses conséquences.

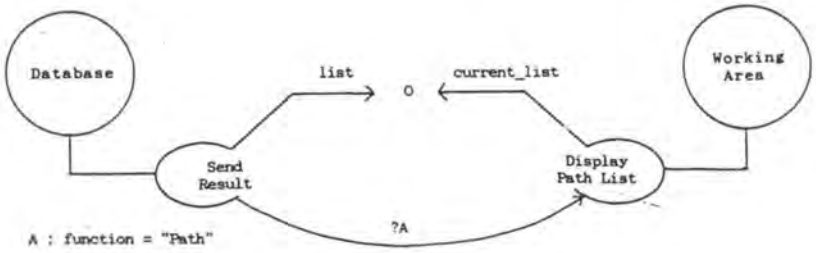


Figure 10.109

Figure 10.110

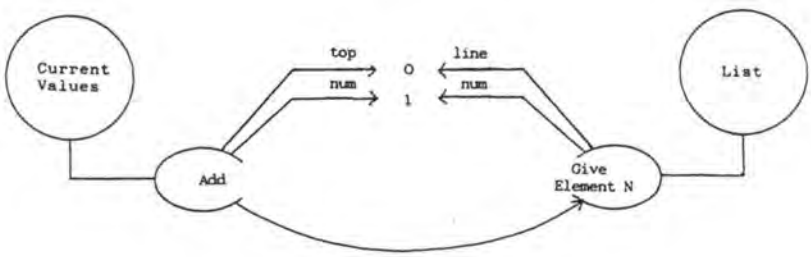


Figure 10.111

La figure 10.112 décrit le retour de l'option (ce qui signifie la mort de la liste courante de la zone de travail et l'abandon de la valeur courante).

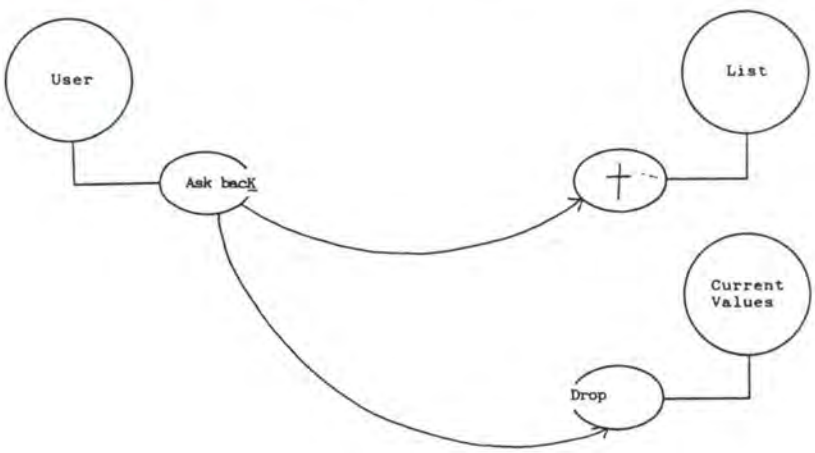


Figure 10.112

L'option Others autorise également des appels aux options Advanced, Basic, Default, Screen, Set-up, Help et Exit. Le lecteur est invité à se reporter aux options respectives. Rappelons que les changements de barre de menu ne sont plus détaillés.

10.4.8 L'option Default

Les figures 10.113 à 10.115 présentent le transfert de la valeur courante dans la valeur par défaut.

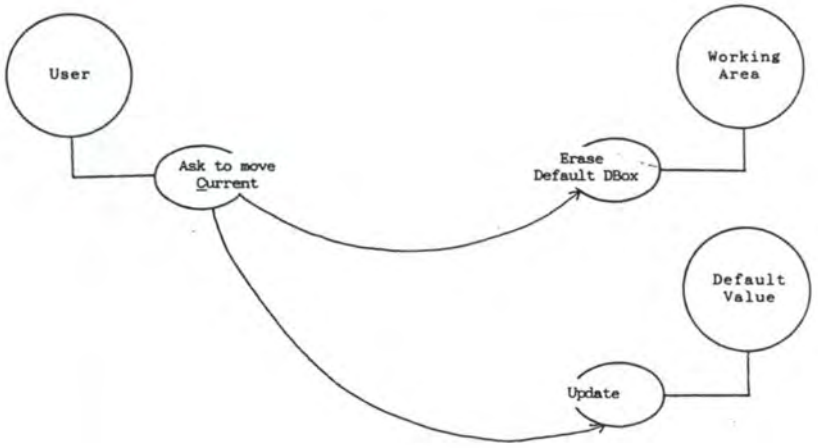


Figure 10.113

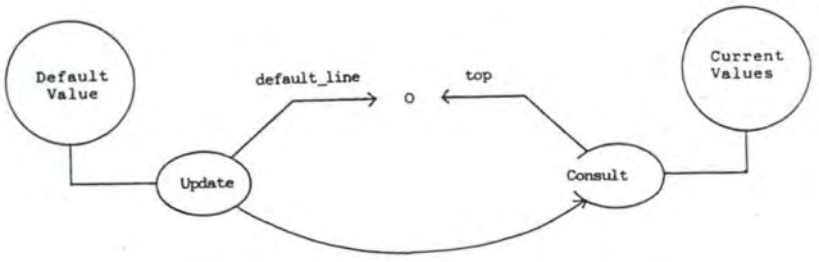


Figure 10.114

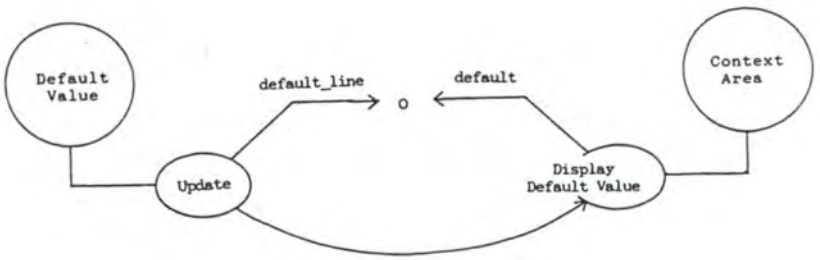


Figure 10.115

Les figures 10.116 et 10.117 présentent le transfert de la valeur par défaut dans la valeur courante.

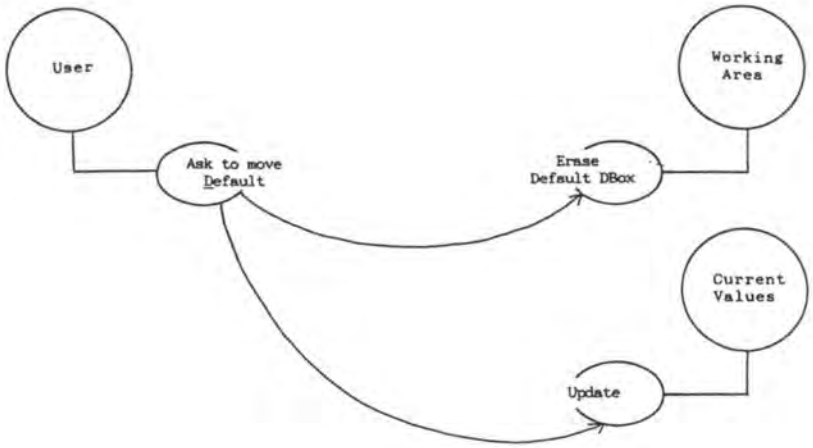


Figure 10.116

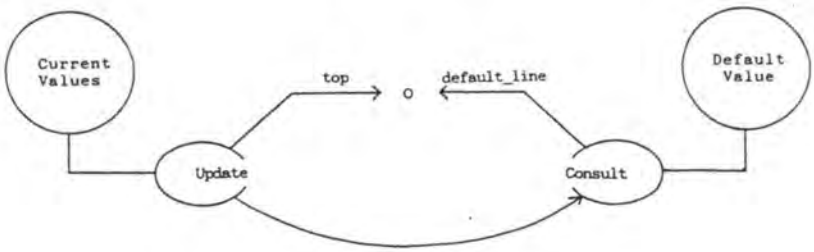


Figure 10.117

L'écran Default autorise également des appels aux options Help et Exit. Le lecteur est invité à se reporter aux options respectives. Rappelons que les changements de barre de menu ne sont plus détaillés.

10.4.9 L'option Set-up

La figure 10.118 décrit le changement de ligne et la figure 10.119 le changement du paramètre de cette ligne.

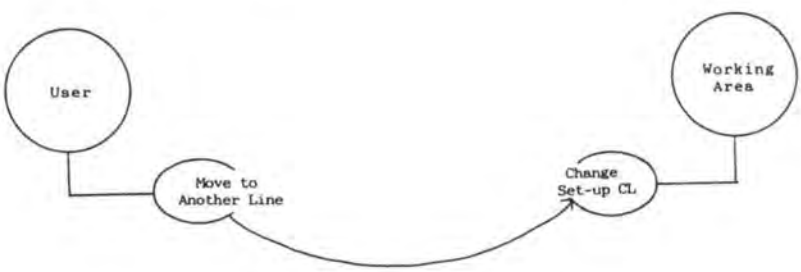


Figure 10.118

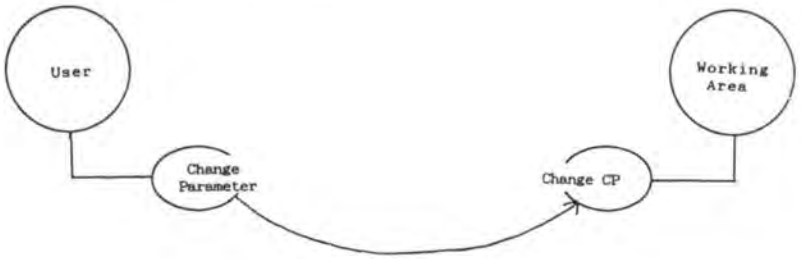


Figure 10.119

L'écran Set-up autorise également des appels aux options Help et Exit. Le lecteur est invité à se reporter aux options respectives. Rappelons que les changements de barre de menu ne sont plus détaillés.

10.4.10 L'option Screen

Les figures 10.120 à 10.123 présentent la mémorisation et le rappel d'un écran. Nous ne nous étendons pas sur l'objet Memorized Screen car, comme nous l'avons déja vu, il est possible que le rappel d'écran ne soit possible que par une mémorisation complexe du contenu de l'écran. Nous avons évité ici de prendre parti pour une implémentation plutôt qu'une autre. Nous nous contentons d'appeler "Copy Screen" l'événement qui représente ce qui est nécessaire pour mémoriser l'écran et "Display Screen Copy" pour sa restitution.

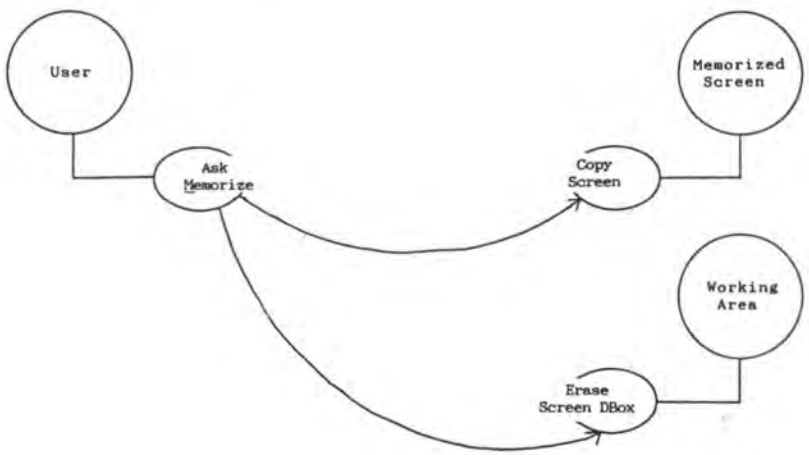


Figure 10.120

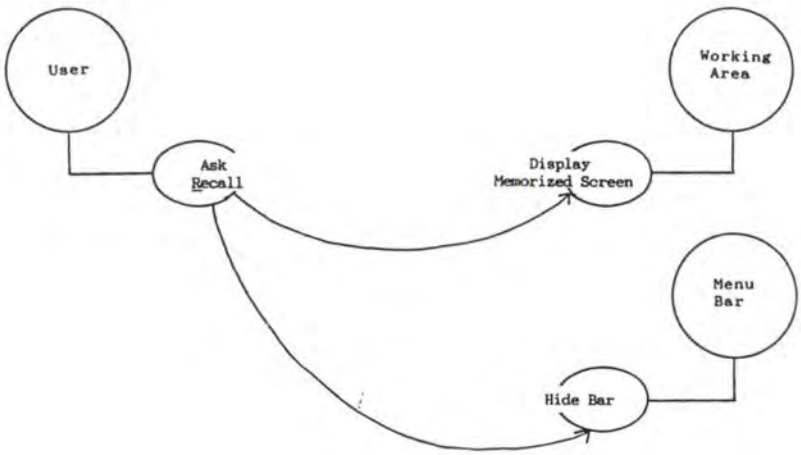


Figure 10.121

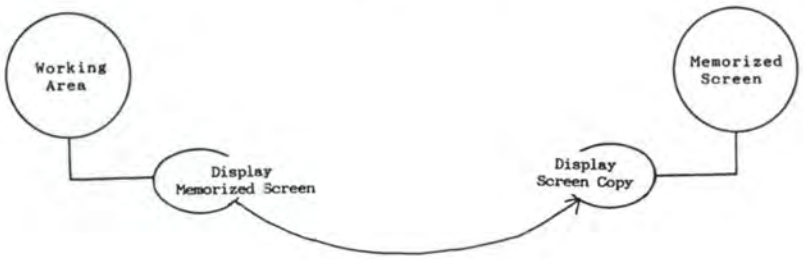


Figure 10.122

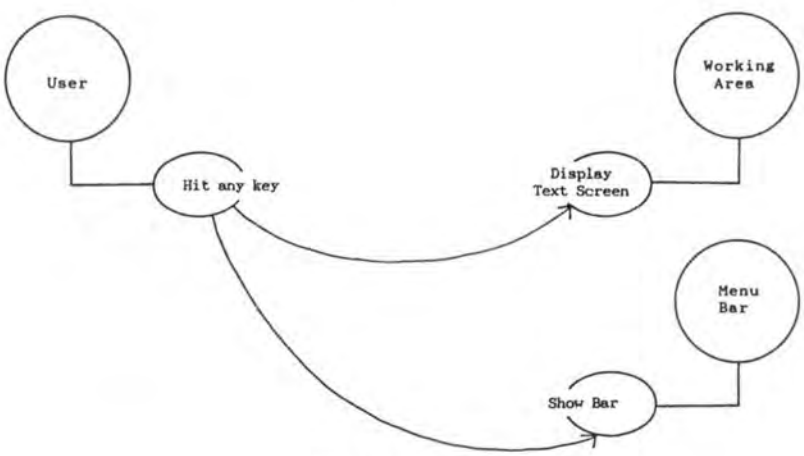


Figure 10.123

L'option Screen autorise également des appels aux options Help et Exit. Le lecteur est invité à se reporter aux options respectives. Rappelons que les changements de barre de menu ne sont plus détaillés.

10.4.11 L'option Help

Les figures 10.124 et 10.125 décrivent l'appel à l'aide et son retour (ici par exemple, dans l'écran Text).

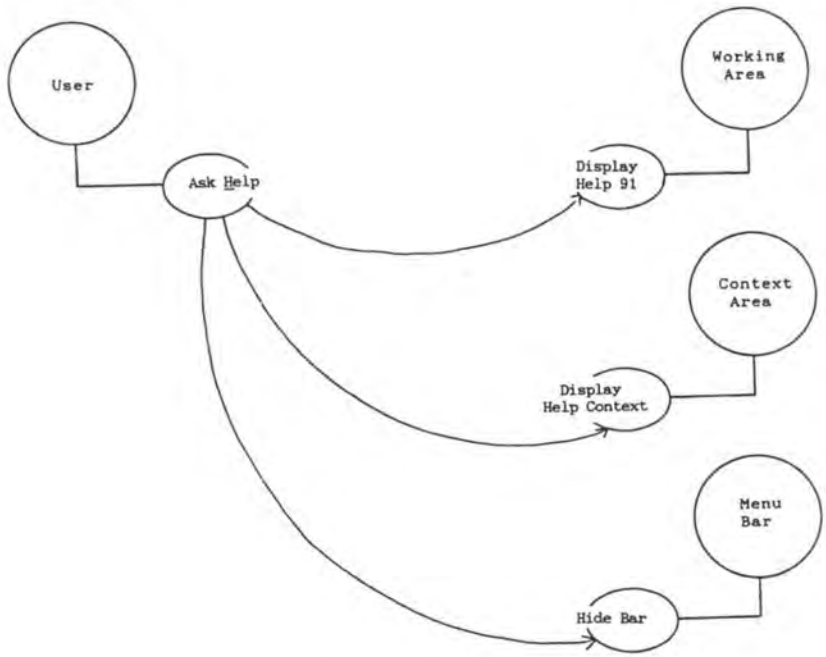


Figure 10.124

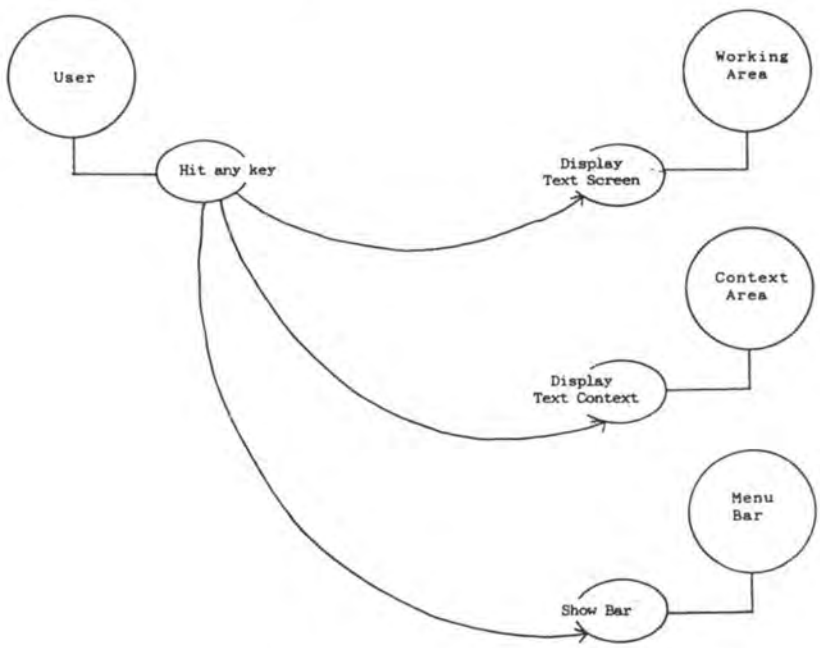


Figure 10.125

10.4.12 La gestion d'erreur

Les figures 10.126 et 10.127 présentent respectivement la survenance d'une erreur dans le cas d'un appel inapproprié (ici par exemple, à List) et dans le cas d'une recherche dans la base de données.

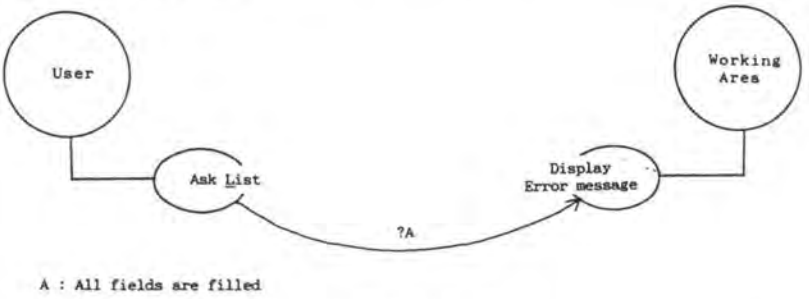


Figure 10.126

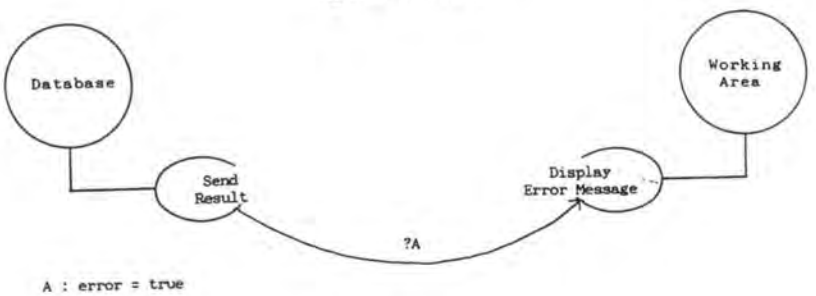


Figure 10.127

La figure 10.128 décrit l'effacement du message d'aide. Rappelons que nous avons choisi de pouvoir effacer le message sans devoir réafficher ce qui se trouvait sur l'écran, comme pour une boîte de dialogue.

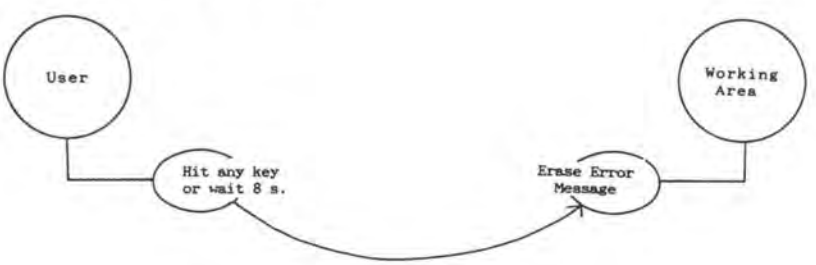


Figure 10.128

10.4.13 La fin de l'application

Rappelons que l'utilisation de l'option "Ask eXit" pour passer de l'écran principal au texte a déjà été présentée.

La figure 10.129 décrit l'appel à la sortie.

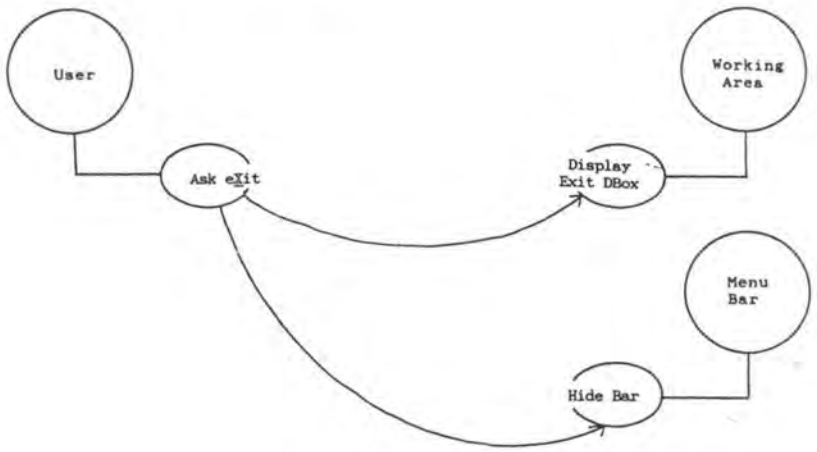


Figure 10.129

La figure 10.130 montre comment abandonner cette sortie.

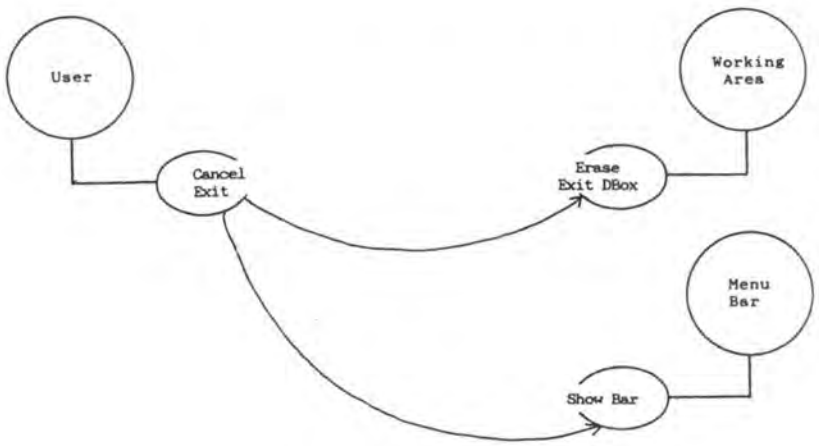


Figure 10.130

Les figures 10.131 à 10.135 présentent la fin de l'application, c'est-à-dire le sauvetage du set-up (figure 10.131 et 10.132) et la mort de tous les objets existants (figure 10.133 à 10.135).

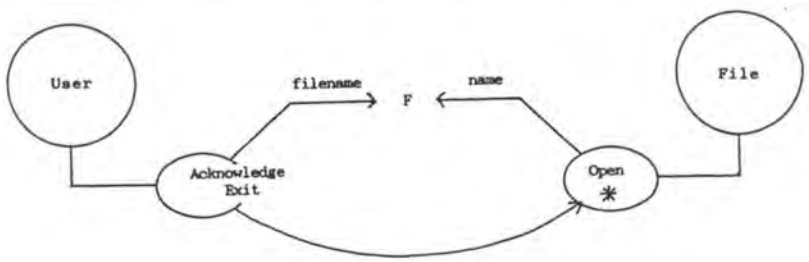


Figure 10.131

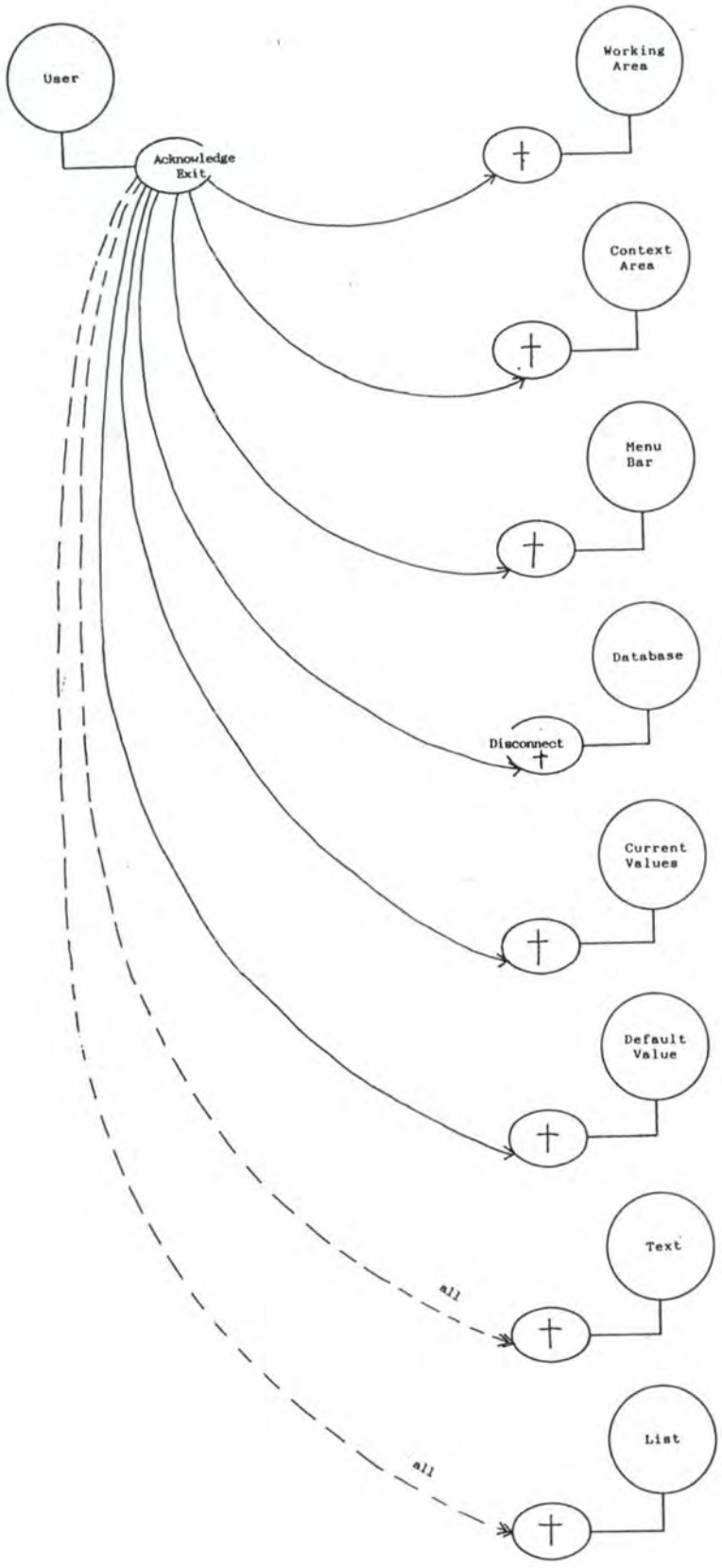


Figure 10.133

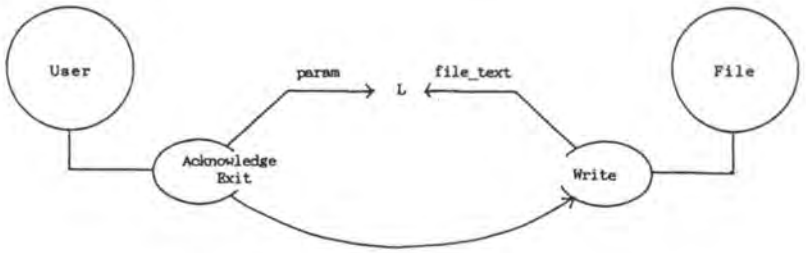


Figure 10.132

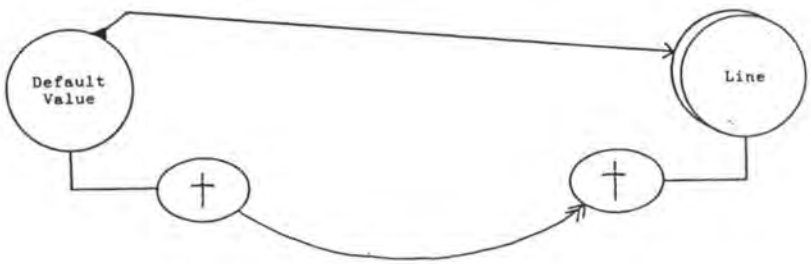


Figure 10.134

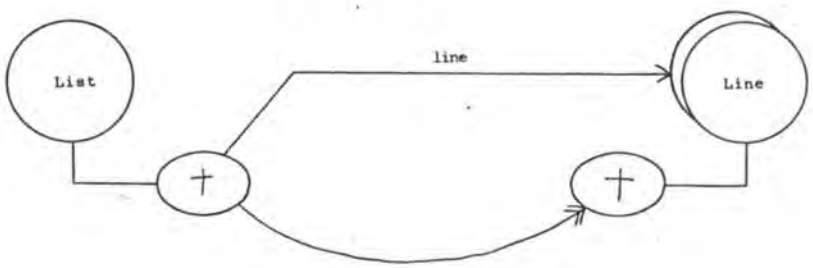


Figure 10.135

10.5 Les diagrammes de mise-à-jour

Rappelons que nous avons choisi de ne présenter que les diagrammes de mise-à-jour qui paraissent apporter une information supplémentaire à celles fournies par les schémas du point précédent.

Les figures 10.136 à 10.139 expriment des mises-à-jour pour l'objet User.

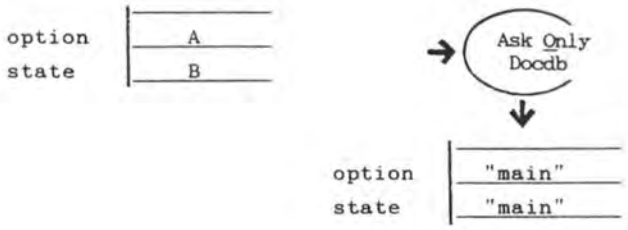


Figure 10.136

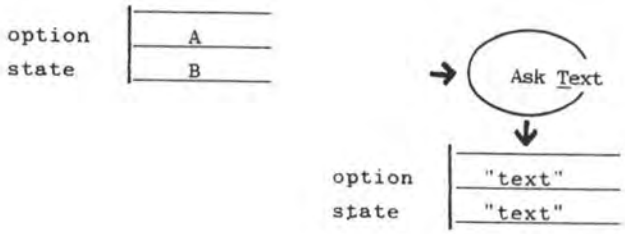


Figure 10.137

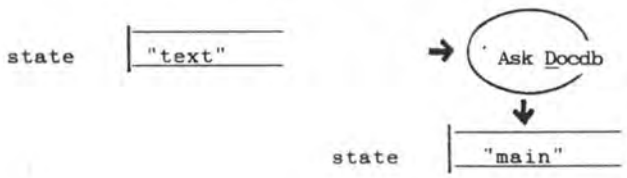


Figure 10.138

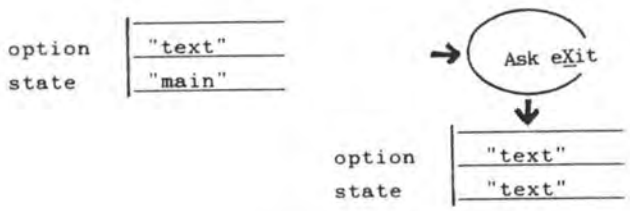


Figure 10.139

Les figures 10.140 à 10.143 décrivent des mises-à-jour de la zone de contexte.

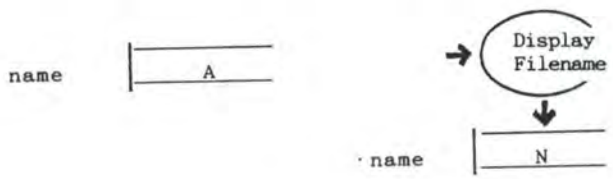


Figure 10.140

Figure 10.141

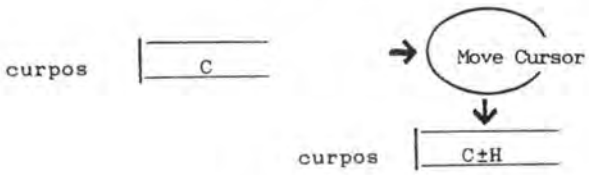


Figure 10.142

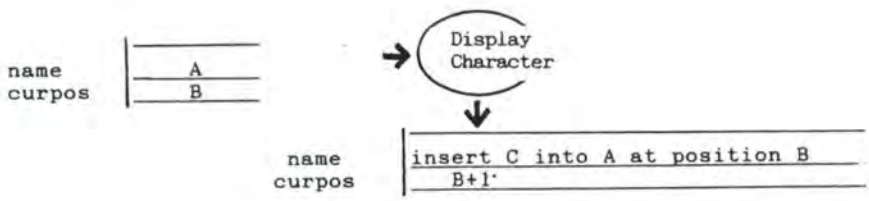


Figure 10.143

La figure 10.144 présente la mise-à-jour de la barre de menu par l'événement "Display Text Bar". Nous n'avons repris sur ce schéma que les slots mis à "True", les slots restants sont mis à "False". Notons qu'il existe quatre barres différentes :

- Initial Bar : Set-up Help Exit (tous disponibles à l'affichage);
- Text Bar : Back Firstpg Lastpg Docdb GetFile Goto Screen Help Set-up Exit (les trois premiers indisponibles à l'affichage);
- Getfile Bar : identique à la Text Bar mais avec uniquement Help Exit disponibles;
- Main Bar : Back List Others Basic Advanced Screen Help Set-up Exit (avec Back indisponible).

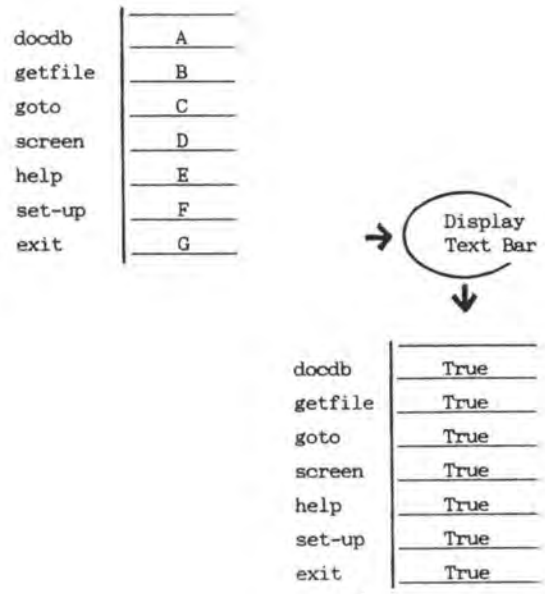


Figure 10.144

Les figures 10.145 et 10.146 décrivent le choix de la fonction d'accès de l'objet Database et la mise-à-jour du slot function (par exemple, si D1V a été choisi).

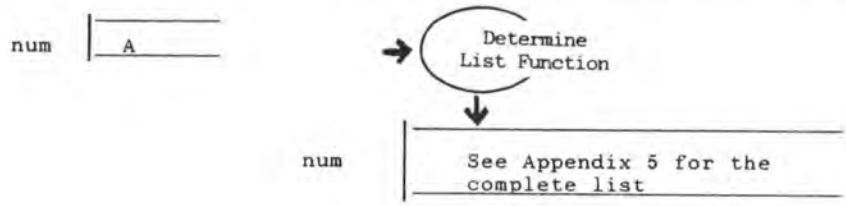


Figure 10.145

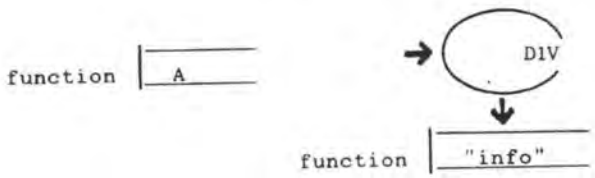


Figure 10.146

Chapitre 11

Modifications des spécifications

Dans ce dernier chapitre, nous présentons trois exemples de modifications. Nous nous sommes inspirés des commentaires issus du feed-back.

Nous envisageons d'abord la possibilité, pour l'utilisateur, de déterminer quels sont les champs résultats dans l'option List. Nous montrons ensuite comment ajouter une nouvelle fonctionnalité dans l'option Others. Enfin, nous décrivons brièvement comment insérer une option d'impression des résultats.

11.1 Sélection des champs de sortie dans l'option List

Supposons d'une part que l'utilisateur dispose d'un moyen pour désigner chacun des champs de l'écran principal. Par exemple sur un terminal texte, la désignation se ferait à l'aide de <CTRL>+1 pour le premier champ.

D'autre part, considérons que le support permet à l'utilisateur de savoir quels champs sont sélectionnés. Par exemple, dans un environnement graphique, des boutons à cocher rempliraient cet office.

Les figures 11.1 et 11.2 montrent ce qu'il faut ajouter au diagramme de comportement respectivement de l'utilisateur et de la zone de travail.

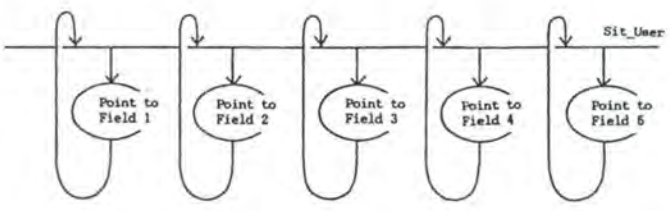


Figure 11.1

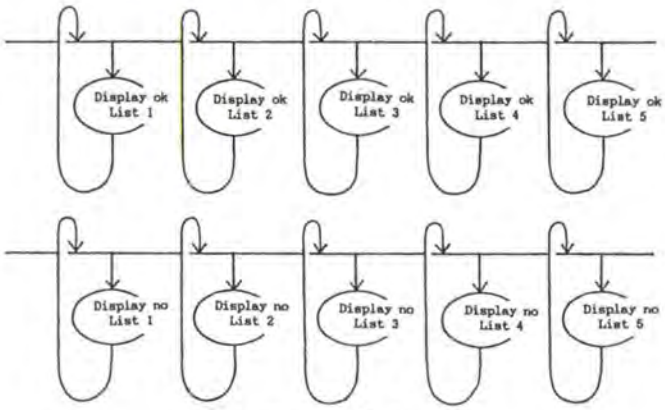


Figure 11.2

Il convient également de modifier le diagramme de comportement de l'objet Database pour y inclure toutes les combinaisons possibles de fonctions d'accès à la place des fonctions de D1V à Y1D (figure 11.3).

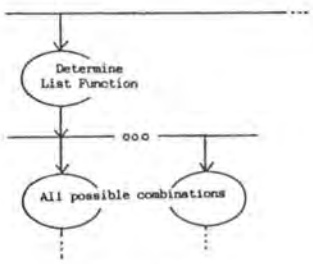


Figure 11.3

Les figures 11.4 et 11.5décrivent les interactions à ajouter.

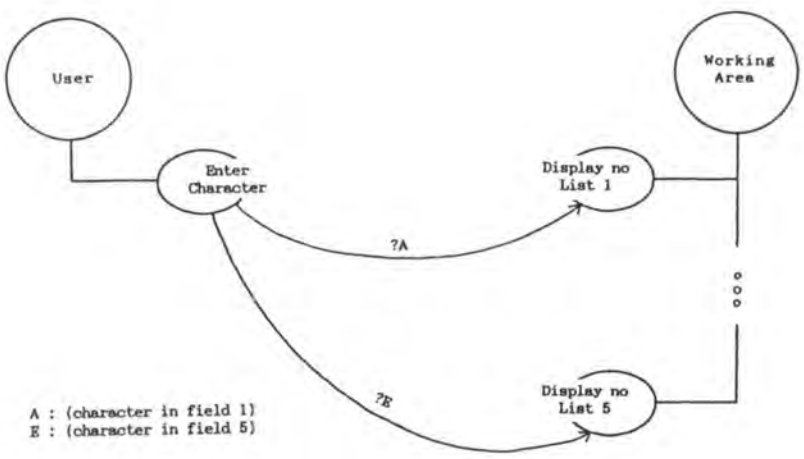


Figure 11.4

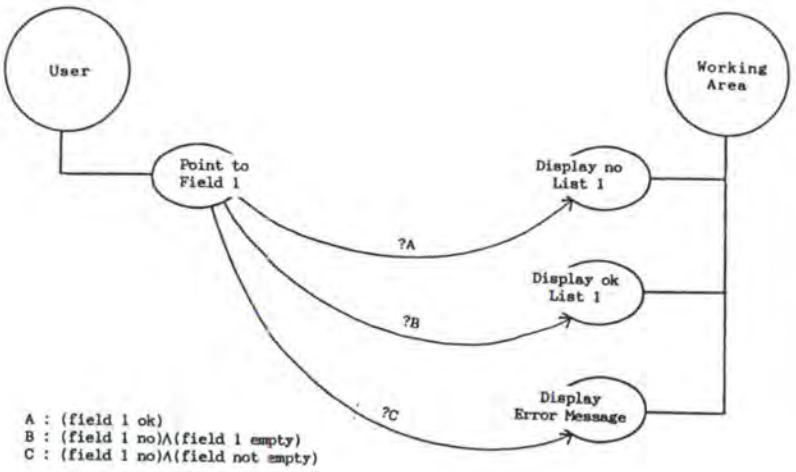


Figure 11.5

11.2 Ajout d'une fonction Others

Bien que cela ne soit pas apparu lors du feed-back, il est possible que de nouvelles fonctions Others s'avèrent nécessaires.

Supposons donc que nous désirions ajouter la fonction X dans le menu Others. Elle accéderait à la base de données via la fonction VLX. Une liste de lignes XLine serait renvoyée comme résultat.

Commençons par souligner qu'aucune modification n'est nécessaire en ce qui concerne l'objet User. Par contre, trois types d'ajouts sont requis pour l'objet Working Area.

Le premier consiste à ajouter dans Others les événements "Display X Screen", "Display X List", "Display X CL" (ainsi que le message d'erreur adéquat). Ces derniers sont enchaînés de la même manière que les événements équivalents de Path (Voir figure 10.32).

La seconde modification revient à reprendre la description de la fonction Path de la figure 10.33 (avec un nouveau numéro).

Le troisième changement consiste à ajouter ce numéro et les affichages nécessaires dans le retour des options Basic & Advanced, Screen, Set-up, Help et Exit (voir respectivement les figures 10.34 à 10.38).

Finalement, il reste à ajouter l'événement VLX dans les diagrammes de l'objet Database (voir figure 10.7 et 10.43) et à compléter les diagrammes d'interaction et d'échange de valeur, notamment la figure 10.108.

11.3 L'option Print

Rappelons que cette option permettrait à l'utilisateur d'imprimer certains résultats.

Nous l'avons modélisée de la façon suivante :

- elle est appelable de partout;
- dans les options List et Others, elle permet d'imprimer soit la liste complète, soit la partie affichée de la liste, soit enfin la valeur courante dans la liste;
- dans une autre situation, l'option Print imprime la zone de travail.

La figure 11.6 présente les modifications apportées au diagramme de comportement de l'objet User.

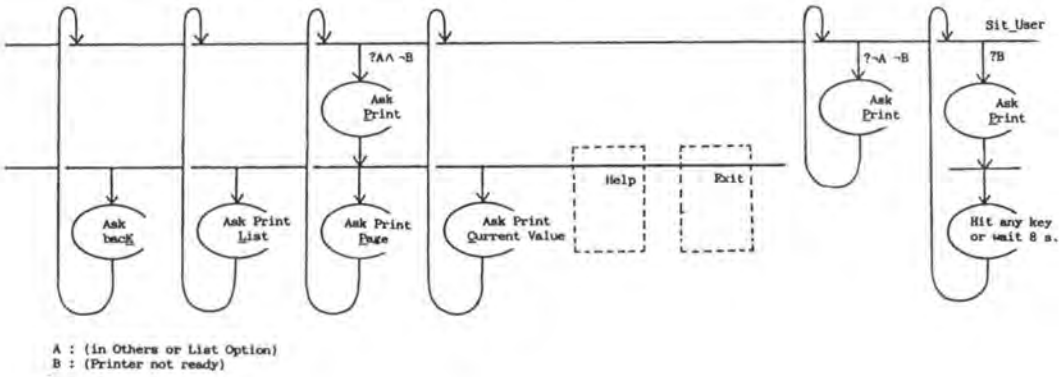


Figure 11.6

La figure 11.7 décrit le comportement du nouvel objet Printer que nous avons défini

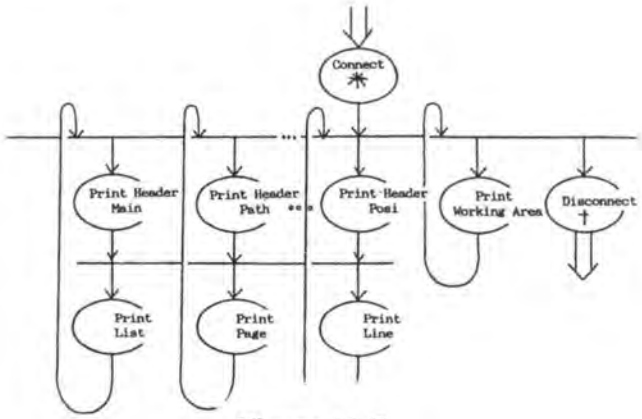


Figure 11.7

Il convient également d'ajouter un événement "Display Print Box" dans la zone de travail et l'option Print dans toutes les barres de menu.

Les figures 11.8 à 11.10 montrent quelques-uns des diagrammes d'interaction pour l'utilisation de cette nouvelle option. La figure 11.9 décrit le cas particulier de l'impression d'une page à partir de l'option List.

Figure 11.8

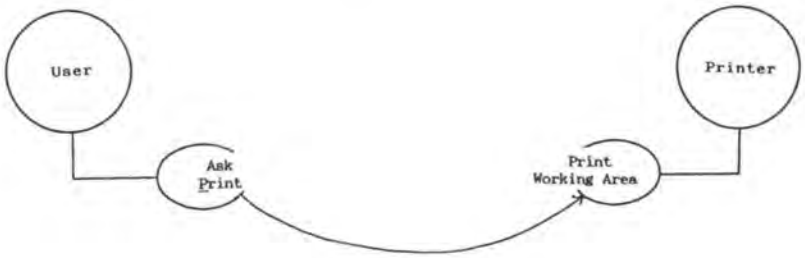


Figure 11.9

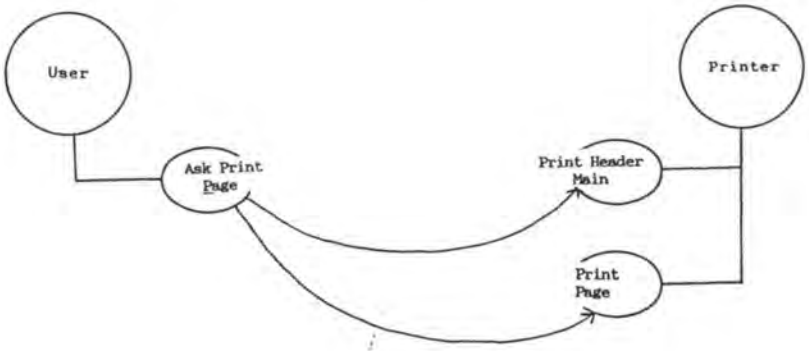


Figure 11.10

Conclusion
et
perspectives d'avenir

Ce mémoire aurait été sans objet sans l'attention que nous avons apportée aux besoins des utilisateurs lors de l'élaboration du prototype. Ce sont eux qui ont déterminé le contenu et la forme de l'application. Mais cette dernière fait partie d'une réalité par définition mouvante et il serait donc utile qu'elle puisse s'adapter le plus facilement possible aux changements.

L'approche orientée objets nécessite un temps d'adaptation lorsque l'on est habitué à penser en termes fonctionnels. Il est cependant indéniable qu'elle permet de limiter le nombre de changements et favorise donc la réutilisabilité.

Même si OBLOG nous a paru un peu lourd pour la spécification d'une application de taille importante, nous devons reconnaître que le fait qu'il soit graphique est séduisant : une fois la syntaxe apprise, il est facile de comprendre les spécifications. Les modifications à y opérer sont aisées (du moins sur le papier). Remarquons que les spécifications doivent être les plus détaillées possible sans toutefois présumer de l'implémentation.

La réalisation de ce mémoire ouvre des perspectives d'avenir. Il nous semble que confronter cette méthode à la pratique serait plein d'enseignements. Notamment, il est intéressant de se demander quels problèmes organisationnels elle pose. En effet, dans la réalité, les analystes ont rarement le temps de rédiger des spécifications complètes et beaucoup travaillent directement à l'étape de programmation. L'approche présentée ici est évidemment absurde si l'implémentation n'est pas précédée de cette phase de spécification.

Un autre problème qui se pose est justement celui du passage des spécifications à l'implémentation. Pour des supports physique et logique donnés, existent probablement des règles de traduction des spécifications, sans lesquelles l'approche resterait "artisanale".

Le développement du prototype et la réflexion sur l'approche que nous avons proposée ont été pour nous une expérience enrichissante. Nous sommes heureux d'avoir pu participer au développement d'un projet aussi prometteur que Docdb.

Bibliographie

[Aupaix91] AUPAIX M.-C., ROELANDT E., *Documentation de bases de données créées lors d'une enquête par questionnaire*, mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique, FUNDP, 1991

[Biggerstaff89] BIGGERSTAFF J., RICHTER C., *Reusability Framework, Assessment and Directions*, in : Software Reusability, Vol.I, Concepts and Models, ACM Press, NY, 1989

[Bodart89] BODART Fr., *Cours introductif aux interfaces Homme-machine*, notes de cours, Institut d'informatique, 1989

[Booch86] BOOCH G., *Object Oriented Development*, in : IEEE Trans. Software Eng., SE-12 (2), 1986

[CES89] CES, *The DRC in the CES - mid-term report*, Centre of Educational Sociology, University of Edinburgh, 1989

[Coad92] COAD P., YOURDON E., *Analyse orientée objets*, Masson, Paris, 1992

[DeMarco79] DE MARCO T., *Structured Analysis and System Specifications*, Prentice-Hall, NJ, 1979

[ESDI92a] ESDI, *About ESDI*, in : Pragmatic Intoduction to the OBLOG Approach in System Design, ESDI, Lisbon, April 1992

[ESDI92b] ESDI, *Object ... what ?*, in : Pragmatic Intoduction to the OBLOG Approach in System Design, ESDI, Lisbon, April 1992

[ESDI92c] ESDI, *OBLOG concepts and their Usefulness*, in : Pragmatic Intoduction to the OBLOG Approach in System design, ESDI, Lisbon, April 1992

[Hainaut86] HAINAUT J.-L.,*Conception assistée des applications informatiques - 2. Conception de la base de données*, Masson-PUN, Paris, 1986

[Lamb91] LAMB J., MIDDLETON L., *Techniques and Software for the Processing of a Large Postal Survey*, in The Statistitian, N°40, 1991, pp. 139-143

[Meyer78] MEYER B., BAUDOIN C., *Méthode de programmation*, Eyrolles, Paris, 1978

[Meyer88] MEYER B., *Object-oriented Software Construction*, Prentice-Hall, NY, 1988

[Nelson91] NELSON M., *An Object Oriented Tower of Babel*, in : OOPS Messenger, 2(3), ACM Press, NY, July 1991

[OVUM90] OVUM, *Reverse engineering*, Ovum ltd, 1990

[Ritchie91] RITCHIE P., *Design of Conceptual Model for Documentation Database and Prototype in SIR*, Centre for Educationnal Sociology, Edinburgh, 1991

[Sommerville89] SOMMERVILLE I., *Software Engineering*, third edition, Addison-Wesley, NY, 1989

[Wirth73] WIRTH N., *Systematic Programming : an introduction*, Prentice-Hall, NJ, 1973

Annexe

1

Base de données DOCDB : description et remplissage

1 Description des tables deDocdb

Extrait de [Ritchie91]

Table DS_DATASETS

COLUMN	LABEL	WIDTH
DATASET	Name of dataset	12
DSTYPE	Type of dataset: ascii, sir, spss etc	8
MACHINE	Name of machine dataset is on	8
FILENAME	Full filename of dataset	32
DSSTATE	State: archive, on-line, tape....	8
SIZE_KB	Size in KB	4
NUMVARS	Number of vars in dset	4
NUMCASES	number of cases in dset	4
ACCESS	Level of security access	8
CASEID	CASEID in Dataset if app.	8
TAPE	Ref name or number of tape	12
TAPELABE	Tape label	12
DSEXPERT	Local expert on dataset	10
DSNOTE	short note on dataset	20
CATNUM	Dataset catalogue number.	16

Table DS_DBRECNAME

COLUMN	LABEL	WIDTH
DATASET	Name of Dataset	12
RECNUM	Number of Record where Variable is located	4
RECNAME	Name of Record where Variable is located	8

Table DS_DBVARRECS

COLUMN	LABEL	WIDTH
VARNAME	Name of Variable	12
DATASET	Name of Dataset	12
RECNUM	Number of Record where Variable is located	4

Table DS_MACHINES

COLUMN	LABEL	WIDTH
MACHNAME	Name of the machine	10
LOCATION	Whereabouts of machine	16
TYPE	Type of machine	16
OPSYS	Operating system of machine	10
MACHNOTE	Additional Notes on machine	60

Table GEN_KEYREFS

COLUMN	LABEL	WIDTH
KERNELID	Survey, Study or dataset ID key	15
REFNUM	Reference number key	4
TITLE	Title of Reference	40
AUTHOR	Author name	30
DATE	Date of Publication	10
PUBLISHER	Published by:	20
REFNOTES	Further notes	80

Table LINK_STDS

COLUMN	LABEL	WIDTH
STUDYID	Unique study identifier	15
DATASET	Name of datasets where study found	12
SELVAR	Var in D'set used to extract study	10
SELVALUE	Value of SELVAR to extract study	4

Table LINK_SUST

COLUMN	LABEL	WIDTH
SURVYID	Short ID of survey	12
STUDYID	Name of (logical) Study within Survey	15

Table ST_QUESTS

COLUMN	LABEL	WIDTH
STUDYID	Unique Study_Identifier	15
DATASET	Name of Dataset	12
QUEST	Questionnaire level	1

Table ST_STUDIES

COLUMN	LABEL	WIDTH
STUDYID	Unique Study Identifier	15
STDYNAME	Full study Name	40
TARGWHO	Who is in study target	20
TARWHERE	Location of target population	10
TARWHEN	Time of sample definition	10
TARSAMP	Target sample of study	4
SAMPFRAC	Sampling Fraction	4
STUDYTYP	Type of study	10
NUMACH	Number of achieved cases in study	4
STDYNOTE	Note on study	30

Table ST_SUBSETS

COLUMN	LABEL	WIDTH
STUDYID	Unique Study_Identifier	15
SUBSETID	Name of subset (eg, Cohort, achieved)	15
SUBSVAR	Var to subset on	8
SUBSVAL	Value of var to subset on	4

Table ST_TEXTNOTES

COLUMN	LABEL	WIDTH
STUDYID	Unique Study Identifier	15
LINENO	Line key no.	4
LINETEXT	Line of text	60

Table ST_WEIGHTS

COLUMN	LABEL	WIDTH
STUDYID	Unique Study Identifier	15
WGHTVAR	Variable to use to weight study	8
COMPVARS	Variable/factor used to derive weighting	8

Table ST_SURVEYS

COLUMN	LABEL	WIDTH
SURVYID	Short ID of survey	12
SVYTITLE	Title of Survey	40
SVYEVENT	Seq # for duplicate despatch	4
DESPDAY	Day of Survey Despatch	4
DESPMTH	Month Of Survey Despatch	4
SVYEAR	Year in which Survey Commenced	4
CLOSEDAY	Day Survey data collection ceased	4
CLOSEMTH	Month Survey data collection stopped	4
CLOSEYR	Year in which Survey closed	4
COMMISSN	Whom survey commissioned by	20
OPERATOR	Who carried out the survey operation	20

Table VI_BASEINFO

COLUMN	LABEL	WIDTH
VARNAME	Name of Variable	10
DATASET	Dataset Identifier	12
VARLAB	Variable Label	40
NMIN	Minimum numeric value of variable	4
NMAX	Maximum numeric value of variable	4
TYPE	Type of Variable	10
SOURCE	Var from quest, derived, admin, SED etc	8
TOPIC	Topic within which Variable falls	10
VARGROUP	Var_Group to which Variable belongs	10
QUESTION	Text of question	100
VAREXP	Local expert on Variable	10
VARNOTE	Note on Variable	10

Table VI_DERIVHOW

COLUMN	LABEL	WIDTH
VARNAME	Name of variable	10
DATASET	Name of dataset	12
COMPVAR	Component variable	10
DERINOTE	Note on derivation for this var	30

Table VI_EQUIVARS

COLUMN	LABEL	WIDTH
DATASET1	Name of 1st dataset	12
VARNAME1	Varname in 1st dataset	10
DATASET2	Name of other dataset	12
VARNAME2	Varname in other dataset	10

Table VI_FILTERVARS

COLUMN	LABEL	WIDTH
VARNAME	Name of variable	10
DATASET	Name of dataset	12

Table VI_LABFREQS

COLUMN	LABEL	WIDTH
VARNAME	Name of variable	8
DATASET	DATASET identifier	12
VARVALUE	Value of variable	4
VALLABEL	Value label of variable	56
ABSFREQ	Absolute frequency in dataset	4
UWPCT	Unweighted percentage	4

Table VI_MISSVALS

COLUMN	LABEL	WIDTH
VARNAME	Name of variable	8
DATASET	DATASET identifier	12
MVAL	Missing_value of variable	4

Table VI_QUAIREPS

COLUMN	LABEL	WIDTH
VARNAME	Name of variable	10
DATASET	Name of dataset	12
QUEST	Name of questionnaire	10
PAGE	Page variable found in questionnaire	4
DEPENDON	Name of var this var depends upon	10

Table VI_RENAMED

COLUMN	LABEL	WIDTH
VARNAME	Name of variable	8
ALIAS	Alternative name of variable	8
DATASET	DATASET identifier	12

Table VI_TOPICS

COLUMN	LABEL	WIDTH
VARNAME	Name of Variable	8
DATASET	Dataset Identifier	12
VARTOPIC	Topic(s) of variable	12

2. Remplissage de la base de données

Table	Nombre de lignes
Bridge	21
Ds_Dataset	17
Ds_Dbrecname	269
Ds_Dbvarrecs	3225
Ds_Machine	3
Gen_Keyrefs	0
Link_Stds	24
Link_Sust	24
St_quests	0
St_Studies	20
St_Subsets	0
St_Textnotes	0
St_Weights	0
Su_Surveys	8
Vi_Baseinfo	10199
Vi_Derivhow	0
Vi_Equivars	472
Vi_Filtervars	89
Vi_Labfreqs	19442
Vi_Missvals	4990
Vi_Quaireps	15547
Vi_Renamed	0
Vi_Topics	3016

Annexe

2

Manuel du "Constraint Checker"

How to use the CONSTRAINT CHECKER

Manual for the programmers

1. Create a text file describing the constraints related to your tabfile.

The program allows you to check for two kind of constraints.

The first kind gives you the possibility to check that a given identifier column is not corrupted (i.e. all the rows are unique for this column). The second one allows you to check that a given column of a table contains only values that are identifiers for another table known as 'identifier reference table'.

Example : the table LINK_STDS has no proper identifier but is made of two columns which held values that are identifiers for two other tables. These tables (ST_STUDIES and DS_DATASETS) are the two "Identifier Reference Tables" for LINK_STDS.

The constraints are written in a particular syntax which is inspired from the LDA language.

The syntax for the first kind of constraints is :

UNIQUE <column name> IN <table name>

Examples : - unique dataset in ds_datasets

- unique srvyid in su_surveys

The syntax for the second kind is :

<col. name>(:<table name>) IN <col. name>(:<table name>)

ex: - dataset(:link_stds) in dataset(:ds_dataset)

- studyid(:link_stds) in studyid(:st_studies)

Note that the second kind of command can be expanded as follow :

C1(:T1) in C2(:T2) and

C3(:T3) in C4(:T4)

with T1=T3 and T2=T4

It allows you to describe constraints such as :

varname(:vi_topics) in varname(:vi_baseinfo) and

dataset(:vi_topics) in dataset(:vi_baseinfo)

You can translate that as : each couple of varname and dataset from vi_topics must exist as a couple in vi_baseinfo

2. Execute the program CONSTRAI.SIR. This is a ICE program which has to be run in DBMS interactive mode.

This program assume the existence of another PQL program called HDLCNSTR. It also assumed the existence of the following PQL procedures: Unparser, Unsqlst, Inparser, Insqlst, Comwrite.

These programs must be in the procedure file. They cannot be stored in a family.

HDLCNSTR does the real job : given the constraints file it translates each constraint into a SQL statement which will performs the attended checking. There is another PQL program called COMPACT which goal is to produce a compact version of the output file from the SQL execution.

When executing CONSTRAIN.SIR you are prompted for several filenames :

- the file which contains the constraints description,
- the one which will hold the resulting SQL statements,
- the name of the .COM file to be produced (see below),
- the name of the tabfile on which the checking will be done,
- and its full filename.

The output files will be :

after running CONSTRAIN.SIR

- <cons_file>.CNS : the file with all the SQL statements to perform the checks.
- <com_file>.COM : a com file which run the checks as a sql batch job.

after running the .COM file

- <out_file>.OUT : the complete output of the SQL session

after running COMPACT

- C<out_file>.OUT : a compact version of the SQL session.

3. When all the files have been produced you must either leave DBMS/IA and run the com file with @<com_file> or run the file from DBMS/IA using the ESCAPE command.

4. Check the content of the .OUT files.

If everything goes well c<out_file>.out is the only file you have to check. It will contain at least one line telling that the tabfile has been properly connected. It will contain other lines only if there are some corrupted identifiers or mismatching with identifier reference tables.

Example Session

An example session is (user input are underlined):

DBMS>CONSTRAI

Constrain description file : unique.cns

SQL statements filename : sql.cns

Com file to produce (no .com extention): new

Name of the tabfile to connect : DOCDB

Tabfile full filename : u4:[ekjc90.docdb]docdb.tfl

DBMS>escape

VMS>@new

VMS>exit

DBMS>compact

Then you must look at the output files (i.e. new.out and cnew.out).

With little chance cnew.out will contain a single line !

Annexe

3

La table Bridge

L'adjonction de la table BRIDGE dans la base de données DOCDB peut se faire à l'aide des commandes SQL suivantes.

- - Create table BRIDGE (survyid char(12), dataset char(12))
  PCTFREE(10)
- - Create unique index BRIDGE_NDX on BRIDGE (survyid asc,
  dataset asc)

Le programme PQL ci-dessous permet de remplir la table.

```
connect tabfile docdb filename 'u4:[ekjc90.docdb]docdb.tfl'
program tupdate
string *12 D,T,SU,TMP
string *15 S,ST
process rows docdb.su_surveys
. get vars SU = survyid
. process rows docdb.link_sust indexed by link_sust via (SU)
. get vars ST = studyid
. process rows docdb.link_stds indexed by link_stds via (ST)
. get vars D  = dataset
. new row is docdb.bridge indexed by bridge_ndx (SU D)
. end row
. end rows
. end rows
end rows
end program
```

Le résulat ressemble à ce qui suit :

```
 TABLE: BRIDGE
   Creation date Dec 03, 1991:23:47:55
   Last Update   Dec 03, 1991:23:50:14
   Update level  5
   Rows          21
   Columns        2
   Indexes        1
   Padding       10
```

Index: BRIDGE_NDX (UNIQUE)
Created on Dec 03, 1991 at 23:49:36.
Number of keys 2
Number of entries 21
Number of levels 1
Padding factor .50
Key 1 : SURVYID
Key 2 : DATASET

Les contraintes pour la nouvelle table sont :

unique survyid dataset in bridge
survyid(:bridge) in survyid(:su_surveys)
dataset(:bridge) in dataset(:ds_datasets)

Annexe

4

Schéma de la base de données INTERFC

```
run name        INTERFC codebook definition
task name       INTERFC codebook initialization commands
old file       INTERFC /create /journal=off
password        ZORGLUB
task name       INTERFC global record type options
max input cols  205
rectype cols    1,2
max rec types   30
max rec count   1023
task name       CIR define case record
n of cases      1000
recs per case   1023
case id         HNUM (A)
common vars      HNUM     (I,3)/
task name       record 1 (SEARCH)  schema definition
record schema   1,SEARCH /lock
document        table used for list produced by 'Search function' (No 1 thru 29
sort ids        COUNT    (A)
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM        3 -   5 (I)
                    /1  COUNT      67 -  70 (I)
                    /1  DATASET     6 -  17 (A)
                    /1  VARNAME    18 -  27 (A)
                    /1  SURVYID    28 -  39 (A)
                    /1  STUDYID    40 -  54 (A)
                    /1  TOPIC      55 -  66 (A)
var labels       HNUM,    'handle : id num for next new user';
                    "/
             COUNT,   'num of line'/
             DATASET,  'id of dataset'/
             VARNAME,  'name of variable'/
             SURVYID,  'id of survey'/
             STUDYID,  'id of study'/
             TOPIC,    'topic of a variable'/
end schema
task name       record 2 (MENU)  schema definition
record schema   2,MENU /lock
document        id. for 'Menu functions' (No 30 to 34)
sort ids        COUNT    (A)
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM        3 -   5 (I)
                    /1  COUNT      67 -  70 (I)
                    /1  VARNAME     6 -  15 (A)
                    /1  DATASET    16 -  27 (A)
                    /1  SURVYID    28 -  39 (A)
                    /1  STUDYID    40 -  54 (A)
                    /1  RECNAME    55 -  62 (A)
                    /1  RECNUM     63 -  66 (I)
var labels       HNUM,    'handle : id num for next new user';
                    "/
             COUNT,   'num of line'/
             VARNAME,  'name of variable'/
             DATASET,  'id of dataset'/
             RECNAME,  'name of a record type where varname is'/
             RECNUM,   'number of the record recname'/
end schema
```

```
task name       record 3 (D1INFO)  schema definition
record schema   3,D1INFO /lock
document        basic informations on a dataset
sort ids        COUNT    (A)
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM        3 -  5 (I)
                    /1  COUNT      76 -  79 (I)
                    /1  DSTYPE      6 -  13 (A)
                    /1  MACHINE    14 -  21 (A)
                    /1  DSSTATE    22 -  29 (A)
                    /1  NUMVARS    30 -  33 (I)
                    /1  NUMCASES   34 -  37 (I)
                    /1  CASEID     38 -  45 (A)
                    /1  DSEXPERT   46 -  55 (A)
                    /1  DSNOTE     56 -  75 (A)
var labels      HNUM,    'handle : id num for next new user';
                    "/
            COUNT,   'num of line'/
            DSTYPE,  'type of dataset'/
            MACHINE, 'name of the machine'/
            DSSTATE, 'state:archive...'/
            CASEID,  'if applicable'/
            DSEXPERT, 'expert for this dataset'/
end schema
task name       record 4 (D2INFO)  schema definition
record schema   4,D2INFO /lock
document        advenced informations on a dataset
sort ids        COUNT    (A)
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM        3 -  5 (I)
                    /1  COUNT      90 -  93 (I)
                    /1  FILENAME    6 -  37 (A)
                    /1  SIZEKB     38 -  41 (I)
                    /1  ACCESS     42 -  49 (A)
                    /1  TAPE       50 -  61 (A)
                    /1  TAPELABE   62 -  73 (A)
                    /1  CATNUM     74 -  89 (A)
var labels      HNUM,    'handle : id num for next new user';
                    "/
            COUNT,   'num of line'/
            TAPE,    'ref. number'/
            TAPELABE, 'label of tape'/
            CATNUM,  'dataset catalogue number'/
end schema
task name       record 5 (S1INFO)  schema definition
record schema   5,S1INFO /lock
document        basic informations about a study
sort ids        COUNT    (A)
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM        3 -  5 (I)
                    /1  COUNT     138 - 141 (I)
                    /1  STDYNAME    6 -  45 (A)
                    /1  TARGWHO    46 -  65 (A)
                    /1  TARWHERE   66 -  75 (A)
                    /1  TARWHEN    76 -  85 (A)
```

```
                /1  TARSAMP    86 -  89 (I)
                /1  SAMPFRAC   90 -  93 (A)
                /1  STUDYTYP   94 - 103 (A)
                /1  NUMACH    104 - 107 (I)
                /1  STDYNOTE  108 - 137 (A)
var labels      HNUM,    'handle : id num for next new user';
                "/
           COUNT,   'num of line'/
           TARSAMP, 'target sample of study'/
           SAMPFRAC, 'sampling fraction'/
           NUMACH,  'number of achieved cases in study'/
end schema
task name       record 6 (U1INFO)  schema definition
record schema   6,U1INFO /lock
document        basic informations about a survey
sort ids        COUNT    (A)
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM       3 -  5 (I)
                /1  COUNT     114 - 117 (I)
                /1  SVYTITLE    6 -  45 (A)
                /1  SVYEVENT   46 -  49 (I)
                /1  DESPDAY    50 -  53 (I)
                /1  DESPMTH    54 -  57 (I)
                /1  SVYEAR     58 -  61 (I)
                /1  CLOSEDAY   62 -  65 (I)
                /1  CLOSEMTH   66 -  69 (I)
                /1  CLOSEYR    70 -  73 (I)
                /1  COMMISSN   74 -  93 (A)
                /1  OPERATOR   94 - 113 (A)
var labels      HNUM,    'handle : id num for next new user';
                "/
           COUNT,   'num of line'/
           SVYEVENT, 'seq # for duplicate despatch'/
           COMMISSN, 'whom survey commissioned by'/
           OPERATOR, 'who carried out the survey operation'/
end schema
task name       record 7 (VLFREQ)  schema definition
record schema   7,VLFREQ /lock
document        frequencies for a given variable
sort ids        COUNT    (A)
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM       3 -  5 (I)
                /1  COUNT      74 -  77 (I)
                /1  VARVALUE    6 -  9 (I)
                /1  VALLABEL   10 -  65 (A)
                /1  ABSFREQ    66 -  69 (I)
                /1  UWPCT      70 -  73 (I)
var labels      HNUM,    'handle : id num for next new user';
                "/
           COUNT,   'num of line'/
           UWPCT,   'unweighted percentage'/
end schema
task name       record 8 (V1INFO)  schema definition
record schema   8,V1INFO /lock
document        basic informations about a variable
sort ids        COUNT    (A)
```

```
sequence check   off
max rec count    1023
data list        fixed (1) /1  HNUM        3 -   5 (I)
                       /1  COUNT      102 -  105 (I)
                       /1  VARLAB       6 -  45 (A)
                       /1  NMIN        46 -  49 (I)
                       /1  NMAX        50 -  53 (I)
                       /1  TYPE        54 -  63 (A)
                       /1  SOURCE      64 -  71 (A)
                       /1  QUESTION   106 - 205 (A)
                       /1  VARGROUP    72 -  81 (A)
                       /1  VAREXP      82 -  91 (A)
                       /1  VARNOTE     92 - 101 (A)
var labels       HNUM,    'handle : id num for next new user';
                       "/
                 COUNT,   'num of line'/
                 VAREXP,  'local expert on variable'/
end schema
task name        record 9 (VLPOSIT)  schema definition
record schema    9,VLPOSIT /lock
document         questionnaire related informations for a variable
sort ids         COUNT    (A)
sequence check   off
max rec count    1023
data list        fixed (1) /1  HNUM        3 -   5 (I)
                       /1  COUNT       34 -  37 (I)
                       /1  QUEST        6 -  15 (A)
                       /1  PAGE        16 -  19 (I)
                       /1  PORDER      20 -  23 (I)
                       /1  DEPENDON    24 -  33 (A)
var labels       HNUM,    'handle : id num for next new user';
                       "/
                 COUNT,   'num of line'/
                 QUEST,   'name of questionnaire'/
                 PORDER,  'order in page NOT IMPLEMENTED'/
                 DEPENDON, 'name of var this var depend on'/
end schema
task name        record 10 (V1REC)  schema definition
record schema    10,V1REC /lock
document         record information for a variable
sort ids         COUNT    (A)
sequence check   off
max rec count    1023
data list        fixed (1) /1  HNUM        3 -   5 (I)
                       /1  COUNT       18 -  21 (I)
                       /1  RECNUM       6 -   9 (I)
                       /1  RECNAME     10 -  17 (A)
var labels       HNUM,    'handle : id num for next new user';
                       "/
                 COUNT,   'num of line'/
end schema
task name        record 11 (VLMISS)  schema definition
record schema    11,VLMISS /lock
document         missing values for a variable
sort ids         COUNT    (A)
sequence check   off
max rec count    1023
data list        fixed (1) /1  HNUM        3 -   5 (I)
```

```
                    /1  COUNT       10 -  13 (I)
                    /1  MVAL        6 -   9 (I)
var labels      HNUM,     'handle : id num for next new user';
                    "/
              COUNT,    'num of line'/
end schema
task name       record 12 (V1FILTR)  schema definition
record schema   12,V1FILTR /lock
document        is a variable filtered
sort ids        COUNT     (A)
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM        3 -   5 (I)
                    /1  COUNT       7 -  10 (I)
                    /1  FILTR       6       (I)
var labels      HNUM,     'handle : id num for next new user';
                    "/
              COUNT,    'num of line'/
              FILTR,    '1 is TRUE; 0 is FALSE'/
end schema
task name       record 13 (MULTIUSE)  schema definition
record schema   13,MULTIUSE /lock
document        MULTIUSER values
sequence check  off
max rec count   1023
data list       fixed (1) /1  HNUM        3 -   5 (I)
                    /1  ATTNUM      6 -   9 (I)
                    /1  LSTLIB      10 -  13 (I)
                    /1  UIN         14 -  17 (I)
var labels      HNUM,     'handle : id num for next new user';
                    "/
              ATTNUM,   'followin number to attribute'/
              LSTLIB,   'last freed'/
              UIN,      'total number of user currently at work'/
end schema
finish
```

Annexe

5

Liste des fonctions d'accès à la base données. Recherches de base

	INPUT					OUTPUT				
	Var	Dst	Std	Svy	Top	Var	Dst	Std	Svy	Top
D1V		+				+				
DS1VY		+	+			+			+	
DST1VY		+	+		+	+			+	
DSTY1V		+	+	+	+	+				
DSY1V		+	+	+		+				
DT1V		+			+	+				
DTY1V		+		+	+	+				
DY1V		+		+		+				
E1D							+			
S1DY			+				+		+	
ST1VDY			+		+	+	+		+	
SY1D			+	+			+			
T1VDSY					+	+	+	+	+	
TY1VDS				+	+	+	+	+		
V1D	+						+			
VD1SY	+	+						+	+	
VDS1TY	+	+	+						+	+
VDSY1T	+	+	+	+						+
VDT1SY	+	+			+			+	+	
VDTY1S	+	+		+	+			+		
VDST1Y	+	+	+		+				+	
VDY1S	+	+		+				+		
VS1DY	+		+				+		+	
VST1DY	+		+		+		+		+	
VSTY1D	+		+	+	+		+			
VSY1D	+		+	+			+			
VT1D	+				+		+			
VTY1D	+		+	+			+			
VY1D	+			+			+			
Y1D				+			+			